P4 Language Tutorial

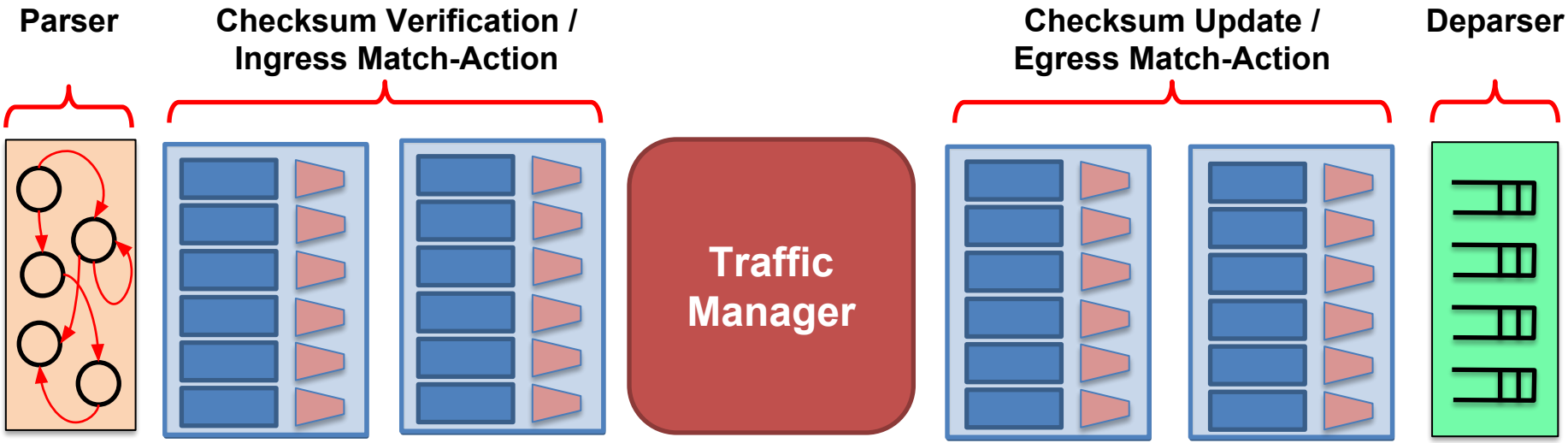
Lab 1: Basics

Before we start...

- **Install VM image (Look for instructor with USB sticks)**
- **Please make sure that your VM is up to date**
  - `$ cd ~/tutorials && git pull`
- **We'll be using several software tools pre-installed on the VM**
  - Bmv2: a P4 software switch
  - `p4c`: the reference P4 compiler
  - Mininet: a lightweight network emulation environment
- **Each directory contains a few scripts**
  - `$ make` : compiles P4 program, execute on Bmv2 in Mininet, populate tables
  - `*.py`: send and receive test packets in Mininet
- **Exercises**
  - Each example comes with an incomplete implementation; your job is to finish it!
  - Look for "TODOs" (or peek at the P4 code in `solution/` if you must)

V1Model Architecture

- **Implemented on top of Bmv2's `simple_switch` target**



V1Model Standard Metadata

```
struct standard_metadata_t {
    bit<9>  ingress_port;
    bit<9>  egress_spec;
    bit<9>  egress_port;
    bit<32> clone_spec;
    bit<32> instance_type;
    bit<1>  drop;
    bit<16> recirculate_port;
    bit<32> packet_length;
    bit<32> enq_timestamp;
    bit<19> enq_qdepth;
    bit<32> deq_timedelta;
    bit<19> deq_qdepth;
    bit<48> ingress_global_timestamp;
    bit<32> lf_field_list;
    bit<16> mcast_grp;
    bit<1>  resubmit_flag;
    bit<16> egress_rid;
    bit<1>  checksum_error;
}
```

- **`ingress_port`** - the port on which the packet arrived
- **`egress_spec`** - the port to which the packet should be sent to
- **`egress_port`** - the port on which the packet is departing from (read only in egress pipeline)

P4$_{16}$ Program Template (V1Model)

```
#include <core.p4>
#include <v1model.p4>
/* HEADERS */
struct metadata { ... }
struct headers {
  ethernet_t   ethernet;
  ipv4_t       ipv4;
}
/* PARSER */
parser MyParser(packet_in packet,
          out headers hdr,
          inout metadata meta,
          inout standard_metadata_t smeta) {
  ...
}
/* CHECKSUM VERIFICATION */
control MyVerifyChecksum(in headers hdr,
                         inout metadata meta) {
   ...
}
/* INGRESS PROCESSING */
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t std_meta) {
  ...
}
```

```
/* EGRESS PROCESSING */
control MyEgress(inout headers hdr,
                 inout metadata meta,
                 inout standard_metadata_t std_meta) {
  ...
}
/* CHECKSUM UPDATE */
control MyComputeChecksum(inout headers hdr,
                          inout metadata meta) {
  ...
}
/* DEPARSER */
control MyDeparser(inout headers hdr,
                   inout metadata meta) {
  ...
}
/* SWITCH */
V1Switch(
  MyParser(),
  MyVerifyChecksum(),
  MyIngress(),
  MyEgress(),
  MyComputeChecksum(),
  MyDeparser()
) main;
```

P4$_{16}$ Hello World (V1Model)

```
#include <core.p4>
#include <v1model.p4>
struct metadata {}
struct headers {}

parser MyParser(packet_in packet,
   out headers hdr,
   inout metadata meta,
   inout standard_metadata_t standard_metadata) {

    state start { transition accept; }
}

control MyVerifyChecksum(inout headers hdr, inout metadata
meta) {   apply {  }   }

control MyIngress(inout headers hdr,
   inout metadata meta,
   inout standard_metadata_t standard_metadata) {
apply {
        if (standard_metadata.ingress_port == 1) {
            standard_metadata.egress_spec = 2;
        } else if (standard_metadata.ingress_port == 2) {
            standard_metadata.egress_spec = 1;
        }
    }
}
```

```
control MyEgress(inout headers hdr,
   inout metadata meta,
   inout standard_metadata_t standard_metadata) {
    apply {  }
}

control MyComputeChecksum(inout headers hdr, inout metadata
meta) {
     apply { }
}

control MyDeparser(packet_out packet, in headers hdr) {
    apply { }
}

V1Switch(
   MyParser(),
   MyVerifyChecksum(),
   MyIngress(),
   MyEgress(),
   MyComputeChecksum(),
   MyDeparser()
) main;
```

P4$_{16}$ Hello World (V1Model)

```
#include <core.p4>
#include <v1model.p4>
struct metadata {}
struct headers {}

parser MyParser(packet_in packet, out headers hdr,
   inout metadata meta,
   inout standard_metadata_t standard_metadata) {
    state start { transition accept; }
}

control MyIngress(inout headers hdr, inout metadata meta,
   inout standard_metadata_t standard_metadata) {
    action set_egress_spec(bit<9> port) {
        standard_metadata.egress_spec = port;
    }
    table forward {
        key = { standard_metadata.ingress_port: exact; }
        actions = {
            set_egress_spec;
            NoAction;
        }
        size = 1024;
        default_action = NoAction();
    }
    apply {   forward.apply();   }
}
```

```
control MyEgress(inout headers hdr,
   inout metadata meta,
   inout standard_metadata_t standard_metadata) {
    apply {  }
}

control MyVerifyChecksum(inout headers hdr, inout metadata
meta) {   apply { }   }

control MyComputeChecksum(inout headers hdr, inout metadata
meta) {   apply { }   }

control MyDeparser(packet_out packet, in headers hdr) {
    apply { }
}

V1Switch( MyParser(), MyVerifyChecksum(), MyIngress(),
MyEgress(), MyComputeChecksum(), MyDeparser() ) main;
```

Key	Action Name	Action Data
1	set_egress_spec	2
2	set_egress_spec	1

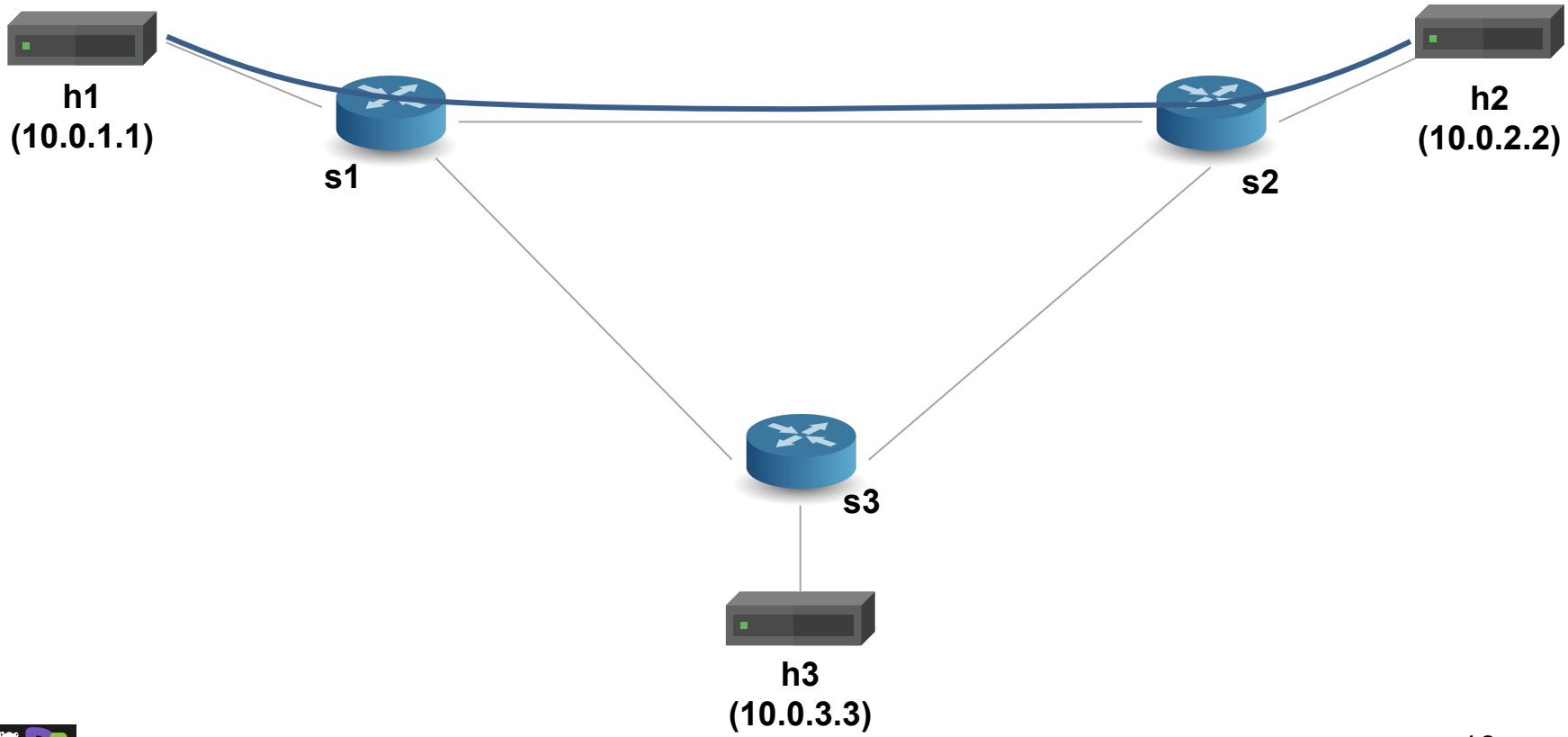
Running Example: Basic Forwarding

- **We'll use a simple application as a running example—a basic router—to illustrate the main features of $P4_{16}$**

- **Basic router functionality:**
  - Parse Ethernet and IPv4 headers from packet
  - Find destination in IPv4 routing table
  - Update source / destination MAC addresses
  - Decrement time-to-live (TTL) field
  - Set the egress port
  - Deparse headers back into a packet

- **We've written some starter code for you (`basic.p4`) and implemented a static control plane**

Basic Forwarding: Topology



$P4_{16}$ Types (Basic and Header Types)

```
typedef bit<48> macAddr_t;
typedef bit<32> ip4Addr_t;
header ethernet_t {
  macAddr_t dstAddr;
  macAddr_t srcAddr;
  bit<16>   etherType;
}
header ipv4_t {
  bit<4>    version;
  bit<4>    ihl;
  bit<8>    diffserv;
  bit<16>   totalLen;
  bit<16>   identification;
  bit<3>    flags;
  bit<13>   fragOffset;
  bit<8>    ttl;
  bit<8>    protocol;
  bit<16>   hdrChecksum;
  ip4Addr_t srcAddr;
  ip4Addr_t dstAddr;
}
```

Basic Types

- **bit<n>**: Unsigned integer (bitstring) of size n
- **bit** is the same as **bit<1>**
- **int<n>**: Signed integer of size n (>=2)
- **varbit<n>**: Variable-length bitstring

Header Types: Ordered collection of members

- Can contain **bit<n>**, **int<n>**, and **varbit<n>**
- Byte-aligned
- Can be valid or invalid
- Provides several operations to test and set validity bit: **isValid()**, **setValid()**, and **setInvalid()**

Typedef: Alternative name for a type

P4$_{16}$ Types (Other Types)

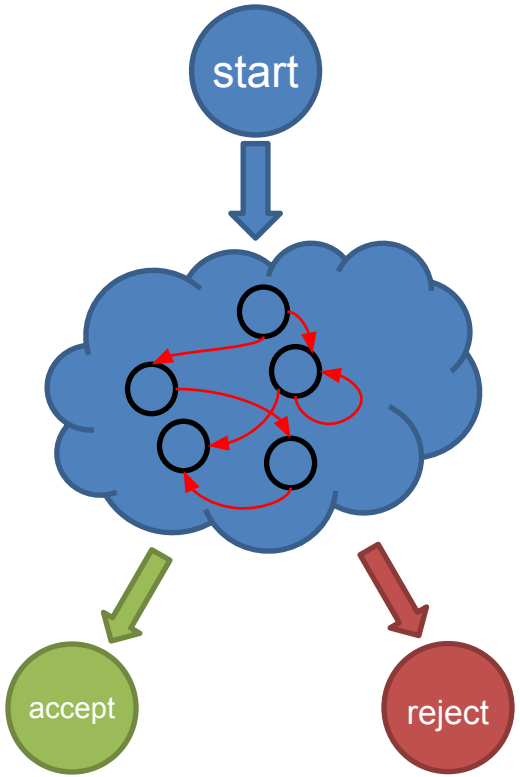
```
/* Architecture */
struct standard_metadata_t {
  bit<9>  ingress_port;
  bit<9>  egress_spec;
  bit<9>  egress_port;
  bit<32> clone_spec;
  bit<32> instance_type;
  bit<1>  drop;
  bit<16> recirculate_port;
  bit<32> packet_length;
  ...
}

/* User program */
struct metadata {
  ...
}
struct headers {
  ethernet_t   ethernet;
  ipv4_t       ipv4;
}
```

Other useful types

- **Struct**: Unordered collection of members (with no alignment restrictions)
- **Header Stack:** array of headers
- **Header Union:** one of several headers

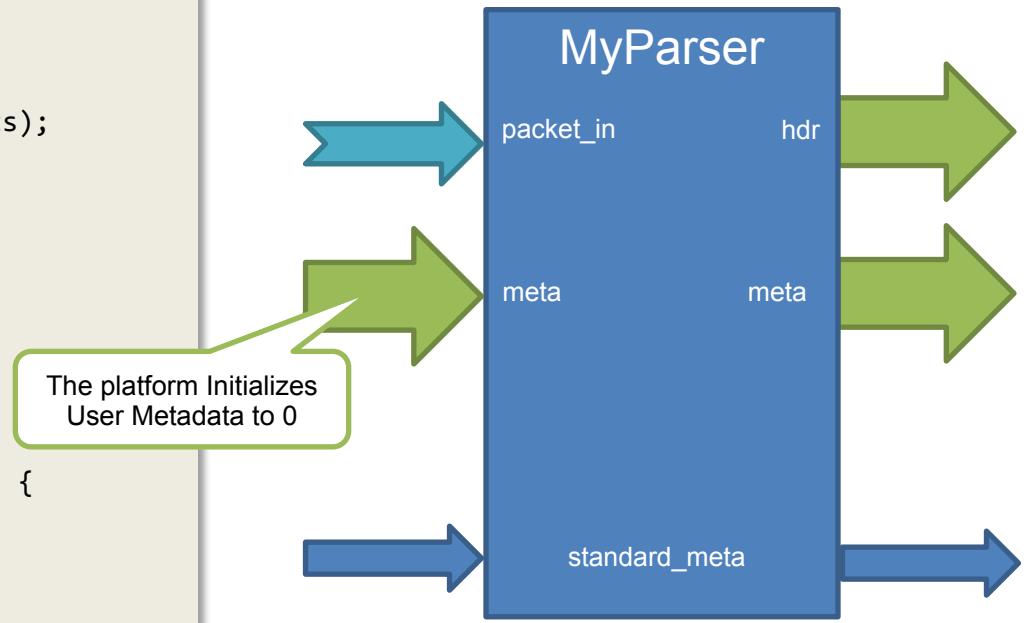
P4$_{16}$ Parsers

- Parsers are functions that map packets into headers and metadata, written in a state machine style
- Every parser has three predefined states
  - `start`
  - `accept`
  - `reject`
- Other states may be defined by the programmer
- In each state, execute zero or more statements, and then transition to another state (loops are OK)



Parsers (V1Model)

```
/* From core.p4 */
extern packet_in {
  void extract<T>(out T hdr);
  void extract<T>(out T variableSizeHeader,
                  in bit<32> variableFieldSizeInBits);
  T lookahead<T>();
  void advance(in bit<32> sizeInBits);
  bit<32> length();
}
/* User Program */
parser MyParser(packet_in packet,
                out headers hdr,
                inout metadata meta,
                inout standard_metadata_t std_meta) {

  state start {
     packet.extract(hdr.ethernet);
     transition accept;
  }

}
```



Select Statement

```
state start {
  transition parse_ethernet;
}

state parse_ethernet {
  packet.extract(hdr.ethernet);
  transition select(hdr.ethernet.etherType) {
    0x800: parse_ipv4;
    default: accept;
  }
}
```

P4$_{16}$ has a `select` statement that can be used to branch in a parser

Similar to `case` statements in C or Java, but without "fall-through behavior"—i.e., `break` statements are not needed

In parsers it is often necessary to branch based on some of the bits just parsed

For example, etherType determines the format of the rest of the packet

Match patterns can either be literals or simple computations such as masks

Coding Break

$P4_{16}$ Controls

- **Similar to C functions (without loops)**
- **Can declare variables, create tables, instantiate externs, etc.**
- **Functionality specified by code in `apply` statement**
- **Represent all kinds of processing that are expressible as DAG:**
  - Match-Action Pipelines
  - Deparsers
  - Additional forms of packet processing (updating checksums)
- **Interfaces with other blocks are governed by user- and architecture-specified types (typically headers and metadata)**

Example: Reflector (V1Model)

```
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t std_meta) {
  bit<48> tmp;
  apply {
    tmp = hdr.ethernet.dstAddr;
    hdr.ethernet.dstAddr = hdr.ethernet.srcAddr;
    hdr.ethernet.srcAddr = tmp;
    std_meta.egress_spec = std_meta.ingress_port;
  }
}
```

Desired Behavior:

- Swap source and destination MAC addresses
- Bounce the packet back out on the physical port that it came into the switch on

Example: Simple Actions

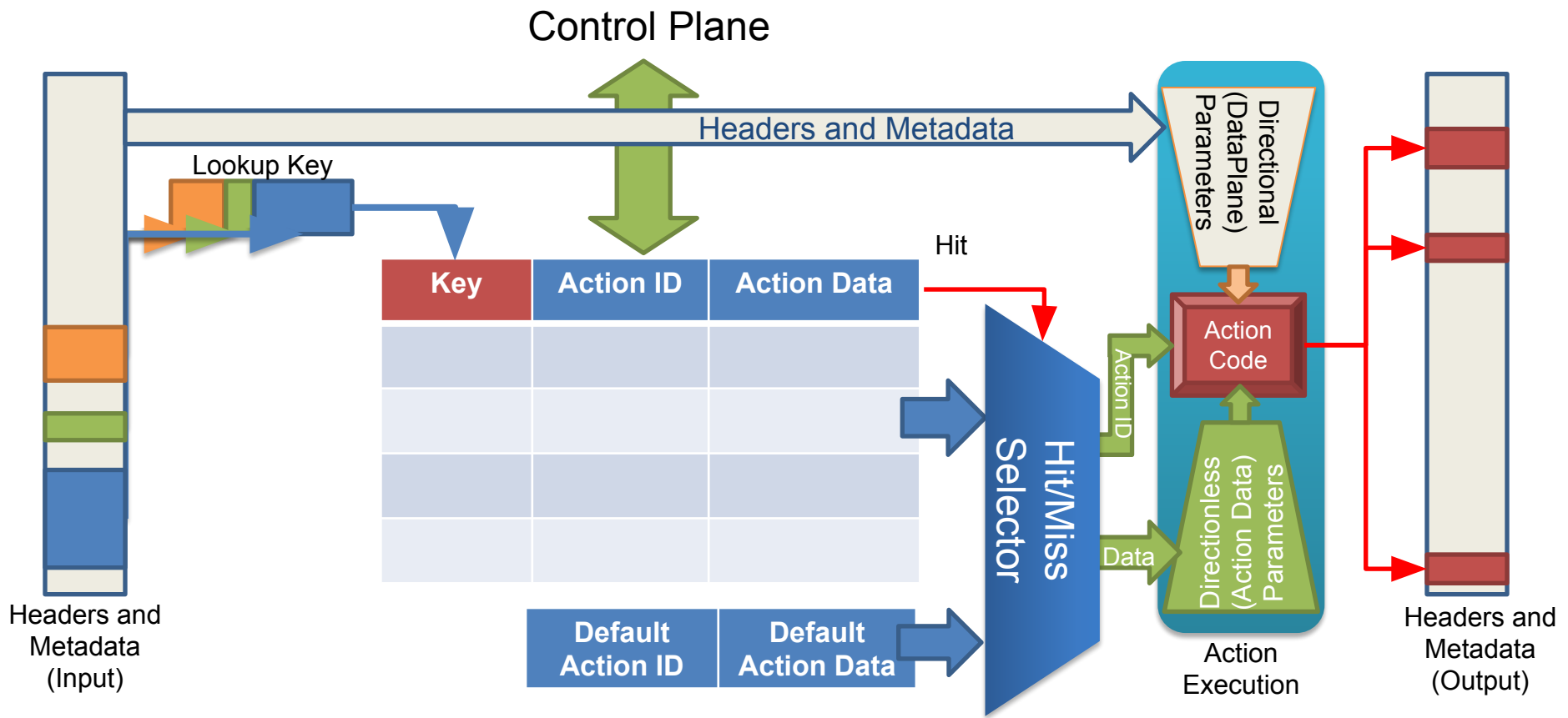
```
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t std_meta) {

  action swap_mac(inout bit<48> src,
                  inout bit<48> dst) {
    bit<48> tmp = src;
    src = dst;
    dst = tmp;
  }

  apply {
    swap_mac(hdr.ethernet.srcAddr,
             hdr.ethernet.dstAddr);
    std_meta.egress_spec = std_meta.ingress_port;
  }
}
```

- **Very similar to C functions**
- **Can be declared inside a control or globally**
- **Parameters have type and direction**
- **Variables can be instantiated inside**
- **Many standard arithmetic and logical operations are supported**
  - +, -, *
  - ~, &, |, ^, >>, <<
  - ==, !=, >, >=, <, <=
  - No division/modulo
- **Non-standard operations:**
  - Bit-slicing: [m:l] (works as l-value too)
  - Bit Concatenation: ++

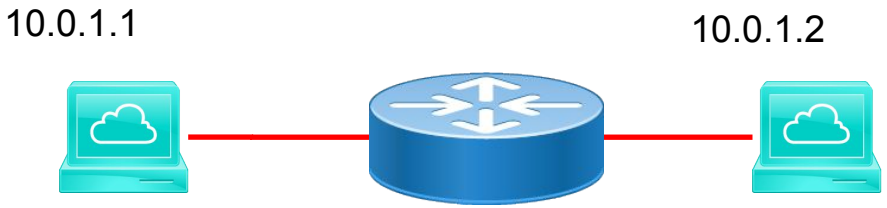
$P4_{16}$ Tables

- **The fundamental unit of a Match-Action Pipeline**
  - Specifies what data to match on and match kind
  - Specifies a list of *possible* actions
  - Optionally specifies a number of table **properties**
    - Size
    - Default action
    - Static entries
    - etc.
- **Each table contains one or more entries (rules)**
- **An entry contains:**
  - A specific key to match on
  - A **single** action that is executed when a packet matches the entry
  - Action data (possibly empty)

Tables: Match-Action Processing



Example: IPv4_LPM Table



Key	Action	Action Data
10.0.1.1/32	ipv4_forward	dstAddr=00:00:00:00:01:01 port=1
10.0.1.2/32	drop	
*`	NoAction	

- **Data Plane (P4) Program**
  - Defines the format of the table
    - Key Fields
    - Actions
    - Action Data
  - Performs the lookup
  - Executes the chosen action
- **Control Plane (IP stack, Routing protocols)**
  - Populates table entries with specific information
    - Based on the configuration
    - Based on automatic discovery
    - Based on protocol calculations

IPv4_LPM Table

```
table ipv4_lpm {
    key = {
      hdr.ipv4.dstAddr: lpm;
    }
    actions = {
      ipv4_forward;
      drop;
      NoAction;
    }
    size = 1024;
    default_action = NoAction();
}
```

Match Kinds

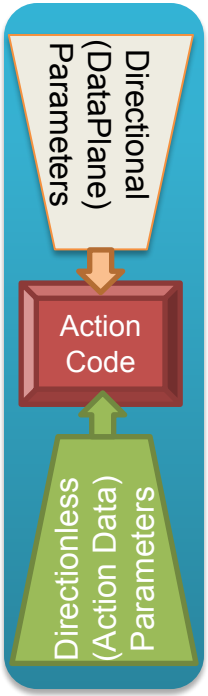
```
/* core.p4 */
match_kind {
    exact,
    ternary,
    lpm
}

/* v1model.p4 */
match_kind {
    range,
    selector
}

/* Some other architecture */
match_kind {
    regexp,
    fuzzy
}
```

- **The type `match_kind` is special in P4**
- **The standard library (`core.p4`) defines three standard match kinds**
  - Exact match
  - Ternary match
  - LPM match
- **The architecture (`v1model.p4`) defines two additional match kinds:**
  - range
  - selector
- **Other architectures may define (and provide implementation for) additional match kinds**

Defining Actions for L3 forwarding

```
/* core.p4 */
action NoAction() {
}

/* basic.p4 */
action drop() {
  mark_to_drop();
}

/* basic.p4 */
action ipv4_forward(macAddr_t dstAddr,
                    bit<9> port) {
  ...
}
```

- **Actions can have two different types of parameters**
  - Directional (from the Data Plane)
  - Directionless (from the Control Plane)
- **Actions that are called directly:**
  - Only use directional parameters
- **Actions used in tables:**
  - Typically use directionless parameters
  - May sometimes use directional parameters too



Action Execution

Applying Tables in Controls

```
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t standard_metadata) {
  table ipv4_lpm {
    ...
  }
  apply {
     ...
     ipv4_lpm.apply();
     ...
  }
}
```

P4$_{16}$ Deparsing

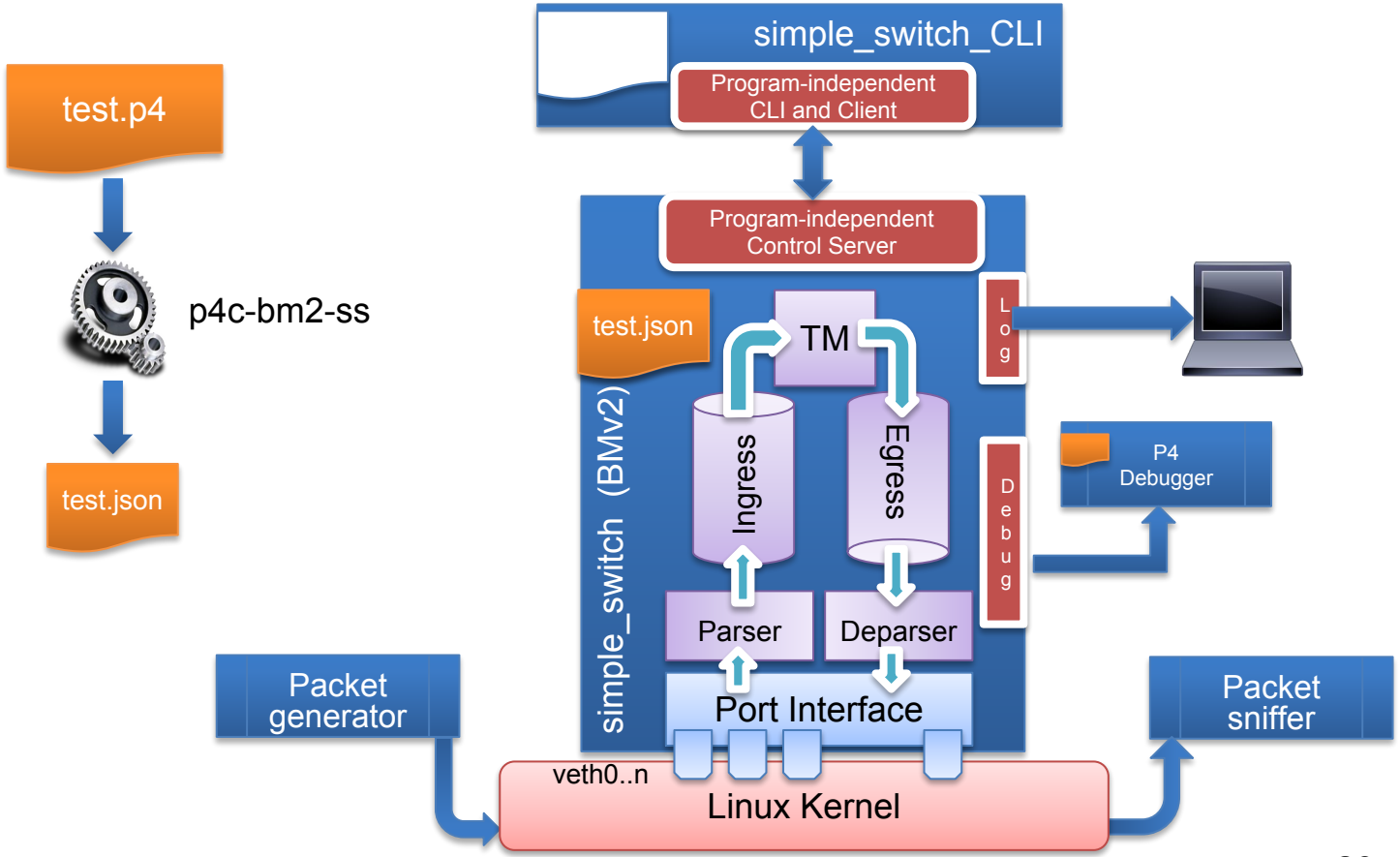
```
/* From core.p4 */
extern packet_out {
  void emit<T>(in T hdr);
}

/* User Program */
control DeparserImpl(packet_out packet,
                     in headers hdr) {
  apply {
    ...
    packet.emit(hdr.ethernet);
    ...
  }
}
```

- **Assembles the headers back into a well-formed packet**
- **Expressed as a control function**
  - No need for another construct!
- **packet_out extern is defined in core.p4:** emit(hdr): serializes header if it is valid
- **Advantages:**
  - Makes deparsing explicit...

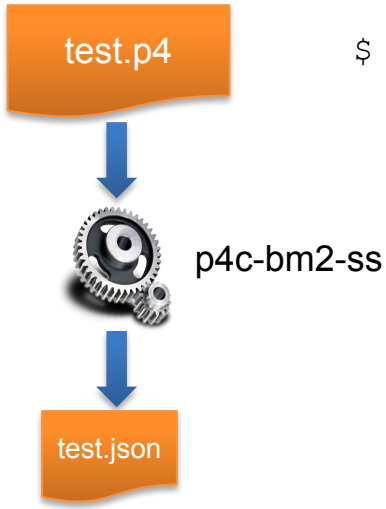
    ...but decouples from parsing

Coding Break

`Makefile`: under the hood

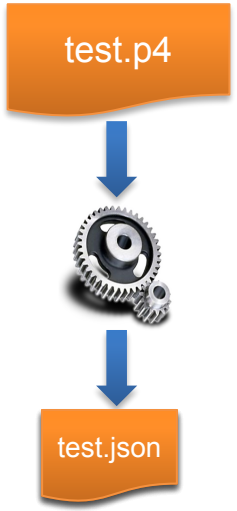


Step 1: P4 Program Compilation

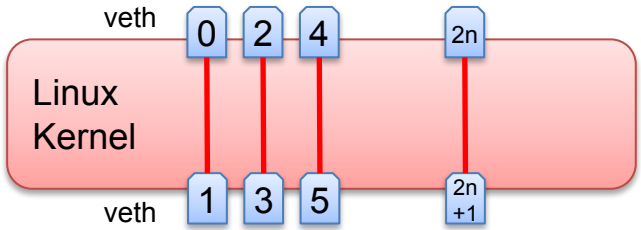



```
$ p4c-bm2-ss -o test.json test.p4
```

Step 2: Preparing veth Interfaces

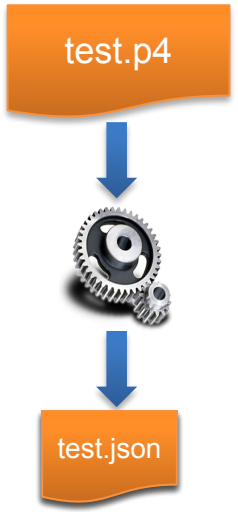


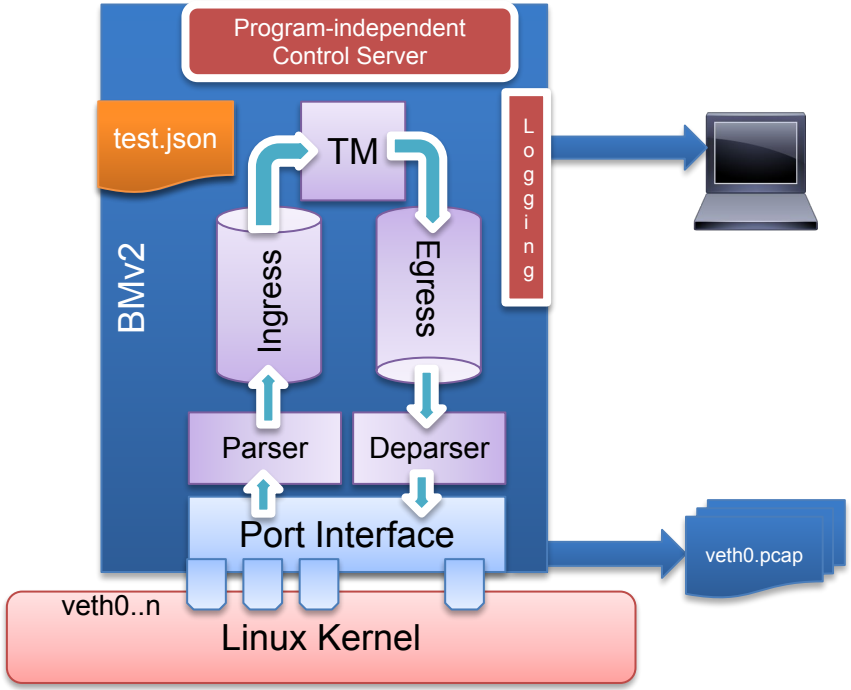

```
$ sudo ~/p4lang/tutorials/examples/veth_setup.sh

# ip link add name veth0 type veth peer name veth1
# for iface in "veth0 veth1"; do
    ip link set dev ${iface} up
    sysctl net.ipv6.conf.${iface}.disable_ipv6=1
    TOE_OPTIONS="rx tx sg tso ufo gso gro lro rxvlan txvlan rxhash"
    for TOE_OPTION in $TOE_OPTIONS; do
        /sbin/ethtool --offload $intf "$TOE_OPTION"
    done
  done
```



Step 3: Starting the model

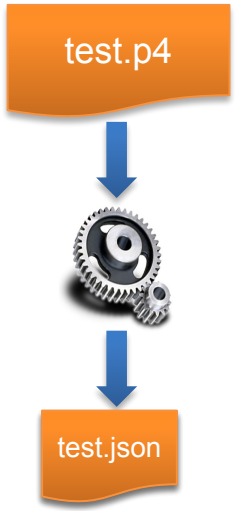
```
$ sudo simple_switch --log-console --dump-packet-data 64 \
       -i 0@veth0 -i 1@veth2 … [--pcap]                  \
       test.json
```

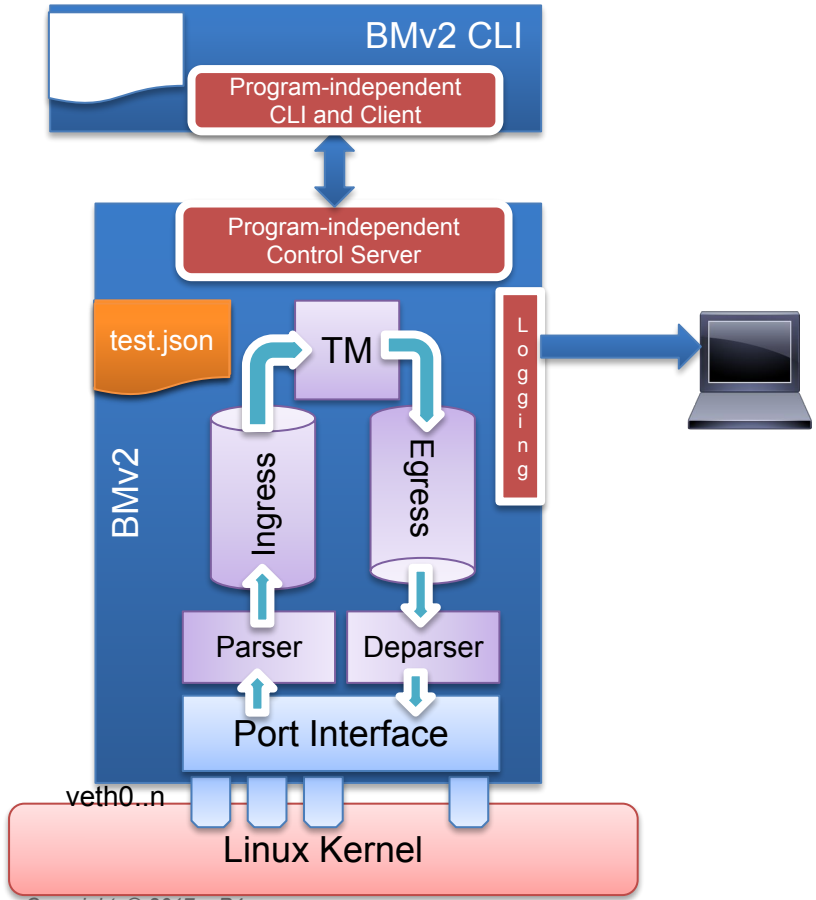




Step 4: Starting the CLI

```
$ simple_switch_CLI
```





Working with Tables in simple_switch_CLI

```
RuntimeCmd: show_tables
m_filter                        [meta.meter_tag(exact, 32)]
m_table                         [ethernet.srcAddr(ternary, 48)]

RuntimeCmd: table_info m_table
m_table                         [ethernet.srcAddr(ternary, 48)]
**********************************************************************
_nop
[]m_action                        [meter_idx(32)]

RuntimeCmd: table_dump m_table
m_table:
0: aaaaaaaaaaaa &&& ffffffffffff => m_action - 0,
SUCCESS

RuntimeCmd: table_add m_table m_action 01:00:00:00:00:00&&&01:00:00:00:00:00 => 1 0
Adding entry to ternary match table m_table
match key:           TERNARY-01:00:00:00:00:00 &&& 01:00:00:00:00:00
action:              m_action
runtime data:        00:00:00:05
SUCCESS
entry has been added with handle 1

RuntimeCmd: table_delete 1
```

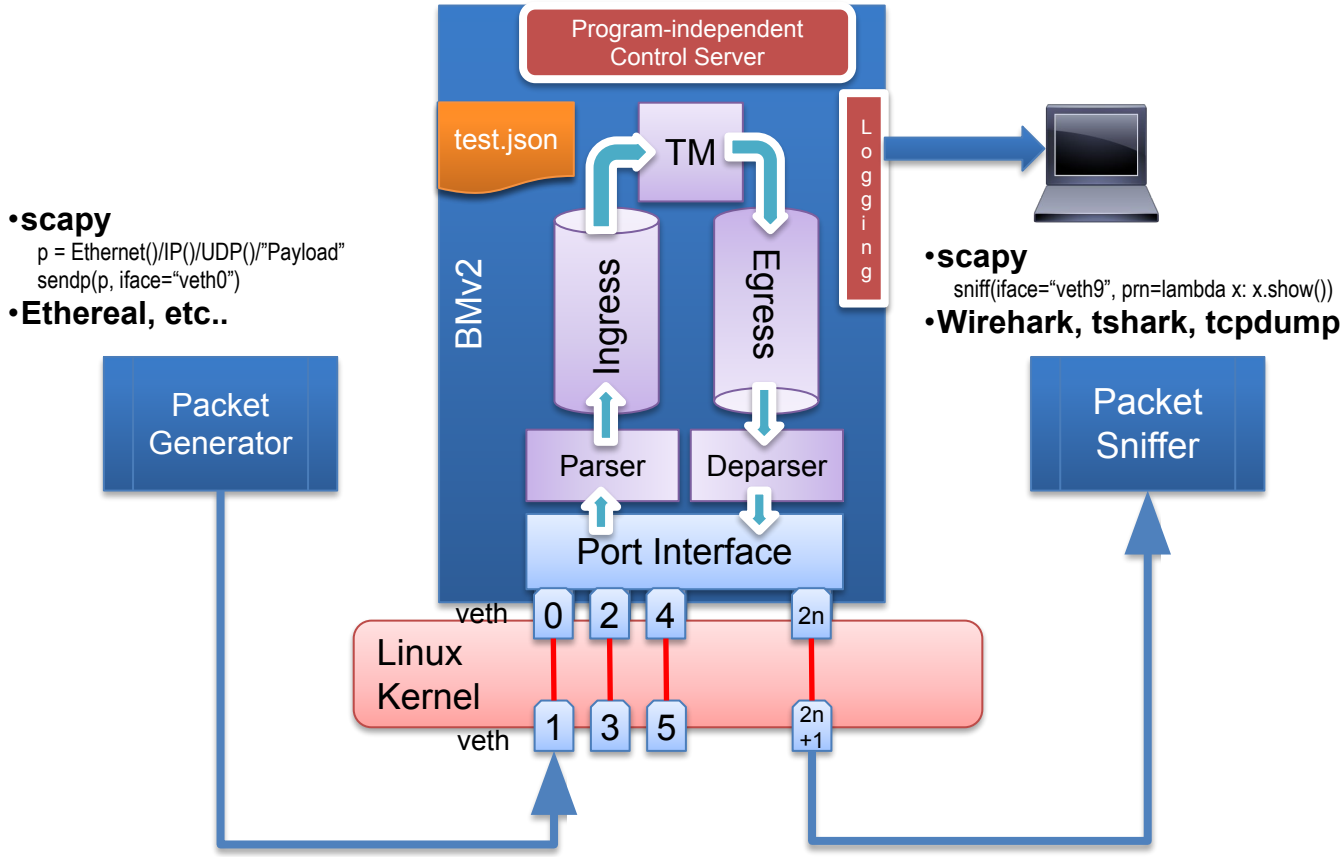
Value and mask for ternary matching. No spaces around "&&&"

Entry priority

"=>" separates the key from the action data

All subsequent operations use the entry handle

Step 5: Sending and Receiving Packets



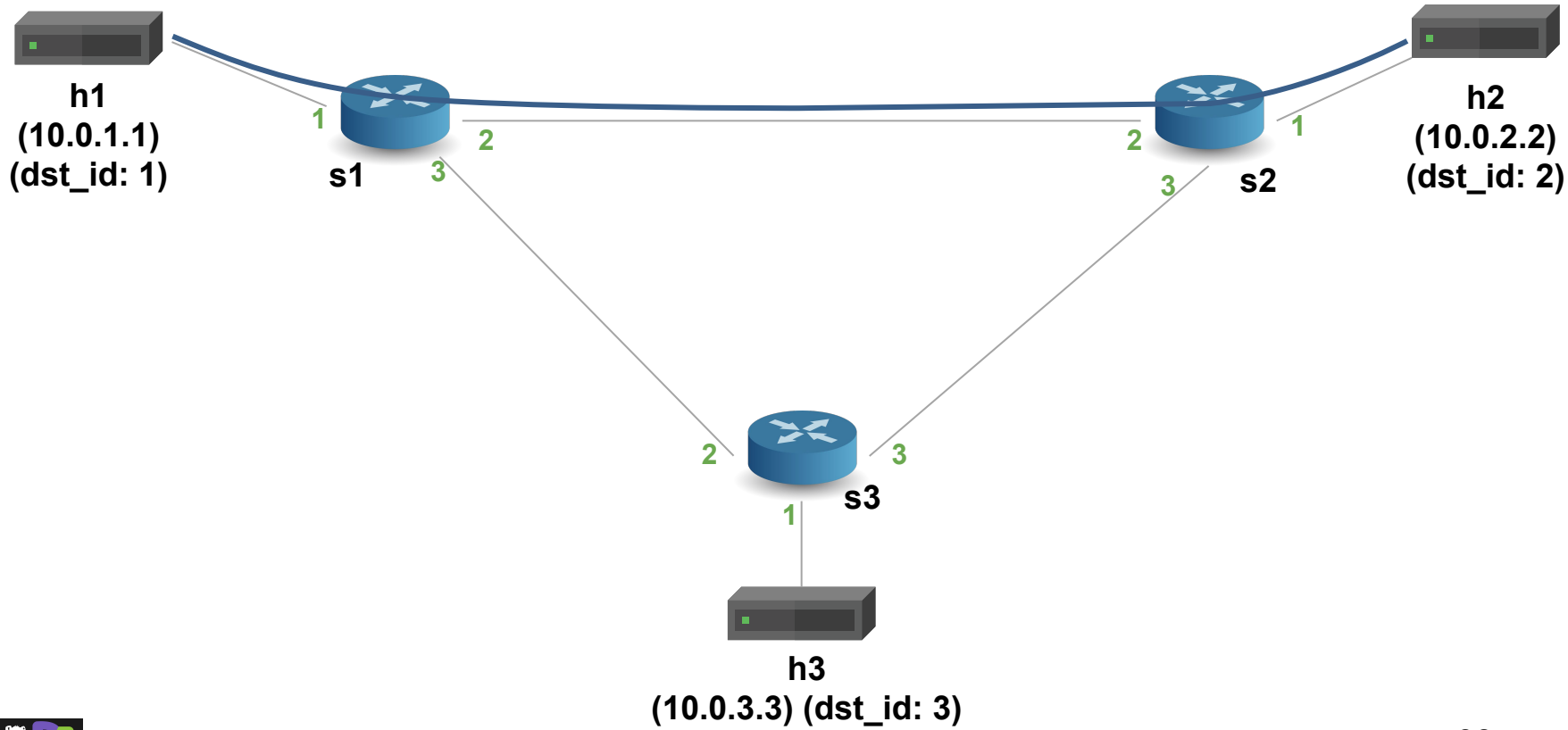
Basic Tunneling

- Add support for basic tunneling to the basic IP router
- Define a new header type (`myTunnel`) to encapsulate the IP packet
- `myTunnel` header includes:
  - `proto_id` : type of packet being encapsulated
  - `dst_id` : ID of destination host
- Modify the switch to perform routing using the `myTunnel` header

Basic Tunneling TODO List

- Define `myTunnel_t` header type and add to `headers` struct
- Update parser
- Define `myTunnel_forward` action
- Define `myTunnel_exact` table
- Update table application logic in `MyIngress` `apply` statement
- Update deparser
- Adding forwarding rules

Basic Forwarding: Topology



Coding Break

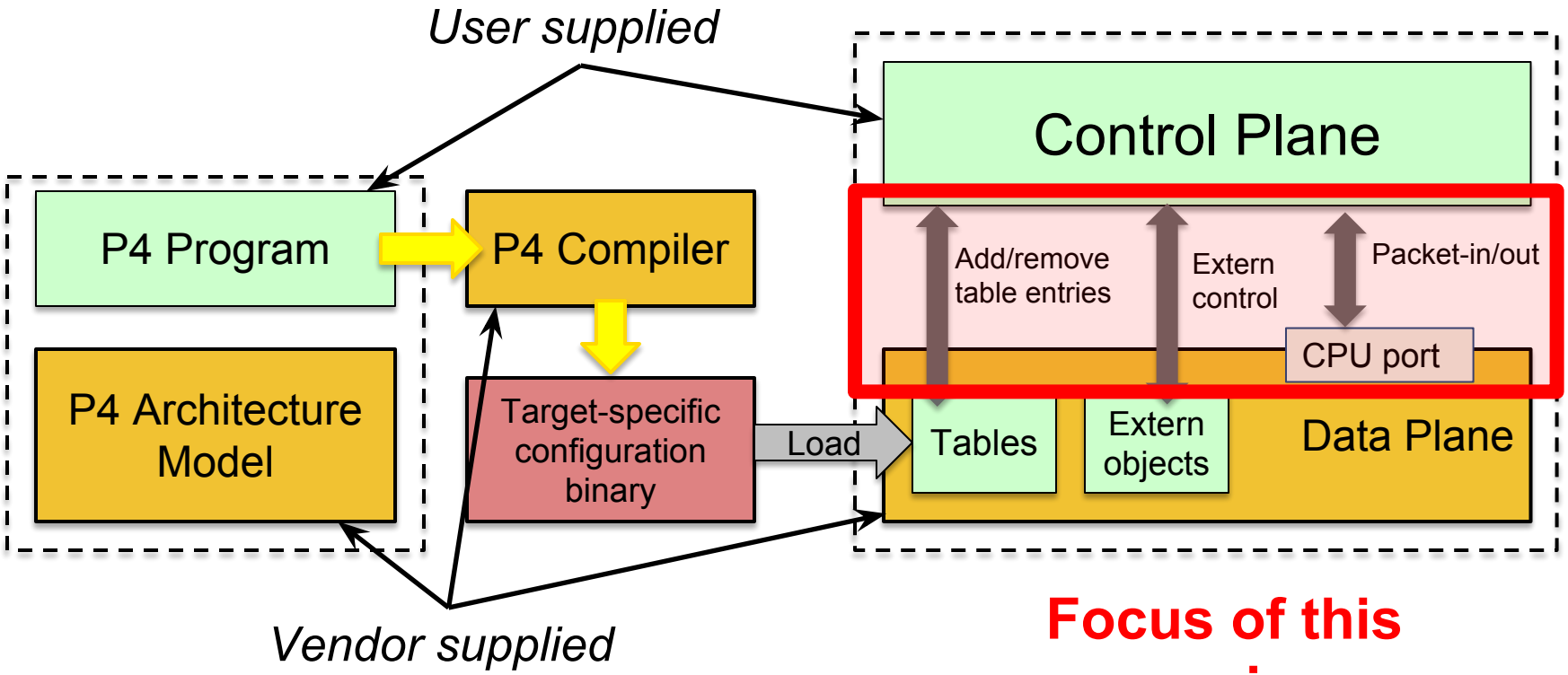
Lab 2: P4 Runtime

Lab 2: P4 Runtime

P4 Runtime

- API overview
- Workflow
- Exercise - Tunneling

Runtime control of P4 data planes



Existing approaches to runtime control

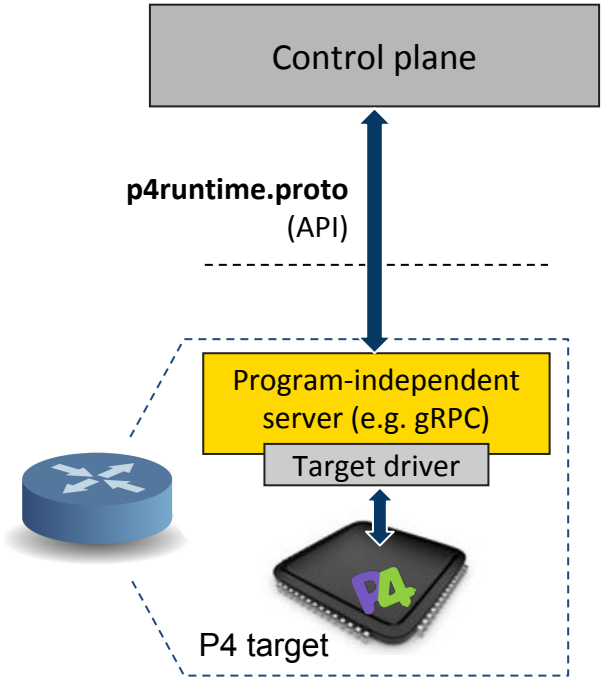
- **P4 compiler auto-generated runtime APIs**
  - Program-dependent -- hard to provision new P4 program without restarting the control plane!
- **BMv2 CLI**
  - Program-independent, but target-specific -- control plane not portable!
- **OpenFlow**
  - Target-independent, but protocol-dependent -- protocol headers and actions baked in the specification!
- **OCP Switch Abstraction Interface (SAI)**
  - Target-independent, but protocol-dependent

Properties of a runtime control API

API	Target-independent	Protocol-independent
P4 compiler auto-generated	✔	✘
BMv2 CLI	✘	✔
OpenFlow	✔	✘
SAI	✔	✘
P4Runtime	✔	✔

What is P4Runtime?

- **Framework for runtime control of P4 targets**
  - Open-source API + server implementation
    - https://github.com/p4lang/PI
  - Initial contribution by Google and Barefoot
- **Work-in-progress by the p4.org API WG**
- **Protobuf-based API definition**
  - p4runtime.proto
  - gRPC as a possible RPC transport
- **P4 program-independent**
  - API doesn't change with the P4 program
- **Enables field-reconfigurability**
  - Ability to push new P4 program without recompiling deployed switches



More details on the P4Runtime API

p4runtime.proto simplified excerpts:

```
message TableEntry {
  uint32 table_id;
  repeated FieldMatch match;
  Action action;
  int32 priority;
  ...
}
```

```
message Action {
  uint32 action_id;
  message Param {
    uint32 param_id;
    bytes value;
  }
  repeated Param params;
}
```

```
message FieldMatch {
  uint32 field_id;
  message Exact {
    bytes value;
  }
  message Ternary {
    bytes value;
    bytes mask;
  }
  ...
  oneof field_match_type {
    Exact exact;
    Ternary ternary;
    ...
  }
}
```

Full protobuf definition:
https://github.com/p4lang/PI/blob/master/proto/p4/p4runtime.proto

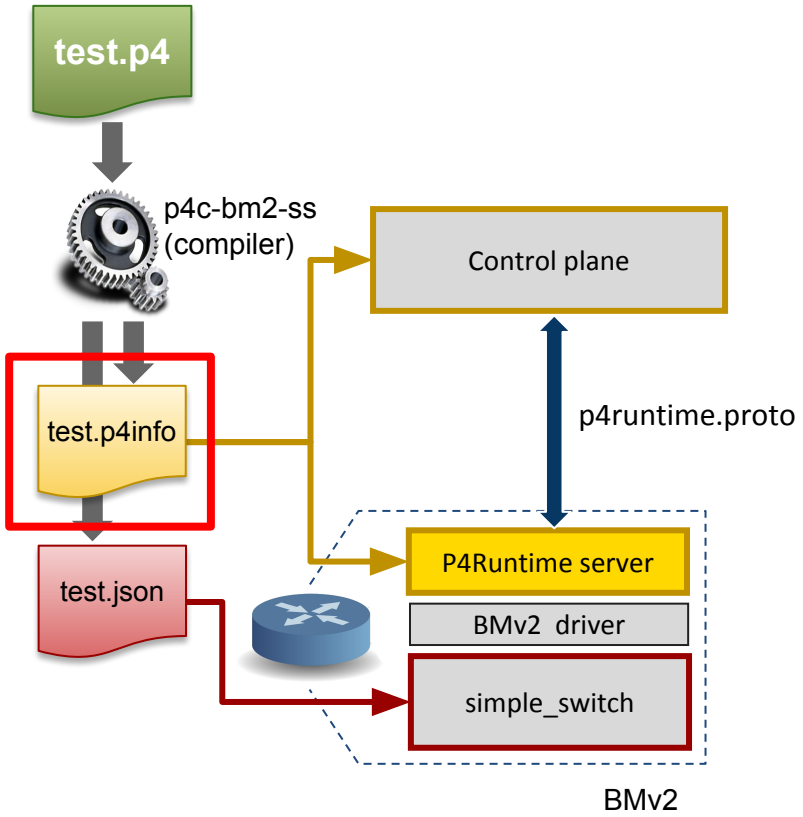
To add a table entry, the control plane needs to know:

- **IDs of P4 entities**
  - Tables, field matches, actions, params, etc.
- **Field matches for the particular table**
  - Match type, bitwidth, etc.
- **Parameters for the particular action**
- **Other P4 program attributes**

P4Runtime workflow

P4Info

- **Captures P4 program attributes needed at runtime**
  - IDs for tables, actions, params, etc.
  - Table structure, action parameters, etc.
- **Protobuf-based format**
- **Target-independent compiler output**
  - Same P4Info for BMv2, ASIC, etc.



Full P4Info protobuf specification:
https://github.com/p4lang/PI/blob/master/proto/p4/config/p4info.proto

P4Info example

basic_router.p4

```
...

action ipv4_forward(bit<48> dstAddr,
                    bit<9> port) {
   /* Action implementation */
}

...

table ipv4_lpm {
   key = {
       hdr.ipv4.dstAddr: lpm;
   }
   actions = {
       ipv4_forward;
       ...
   }
   ...
}
```

P4 compiler

basic_router.p4info

```
actions {
  id: 16786453
  name: "ipv4_forward"
  params {
    id: 1
    name: "dstAddr"
    bitwidth: 48
    ...
    id: 2
    name: "port"
    bitwidth: 9
  }
}
...
tables {
  id: 33581985
  name: "ipv4_lpm"
  match_fields {
    id: 1
    name: "hdr.ipv4.dstAddr"
    bitwidth: 32
    match_type: LPM
  }
  action_ref_id: 16786453
}
```

P4Runtime example

basic_router.p4

```
action ipv4_forward(bit<48> dstAddr,
                    bit<9>  port) {
   /* Action implementation */
}
table ipv4_lpm {
   key = {
       hdr.ipv4.dstAddr: lpm;
   }
   actions = {
       ipv4_forward;
       ...
   }
   ...
}
```

Control plane generates

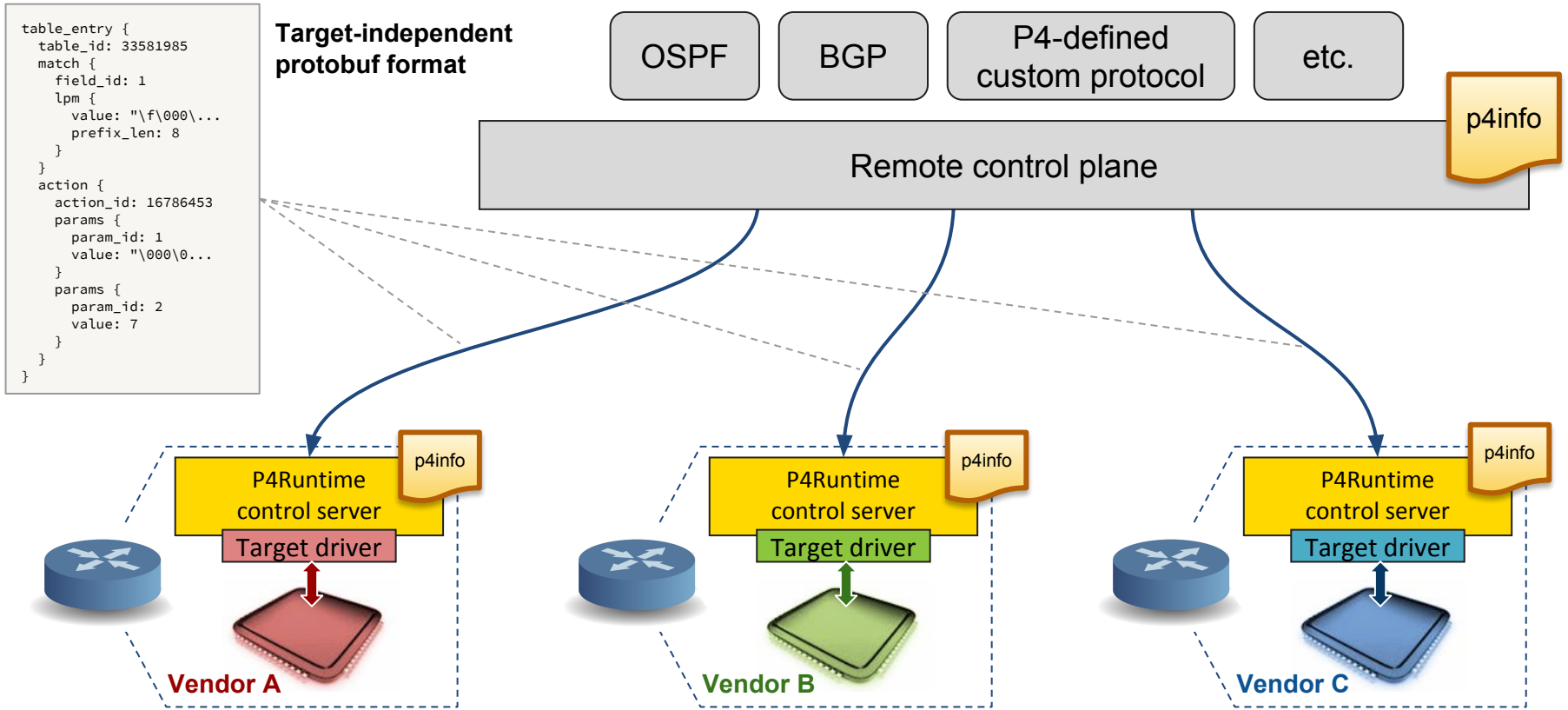
Logical view of table entry

```
hdr.ipv4.dstAddr=10.0.1.1/32
      -> ipv4_forward(00:00:00:00:00:10, 7)
```

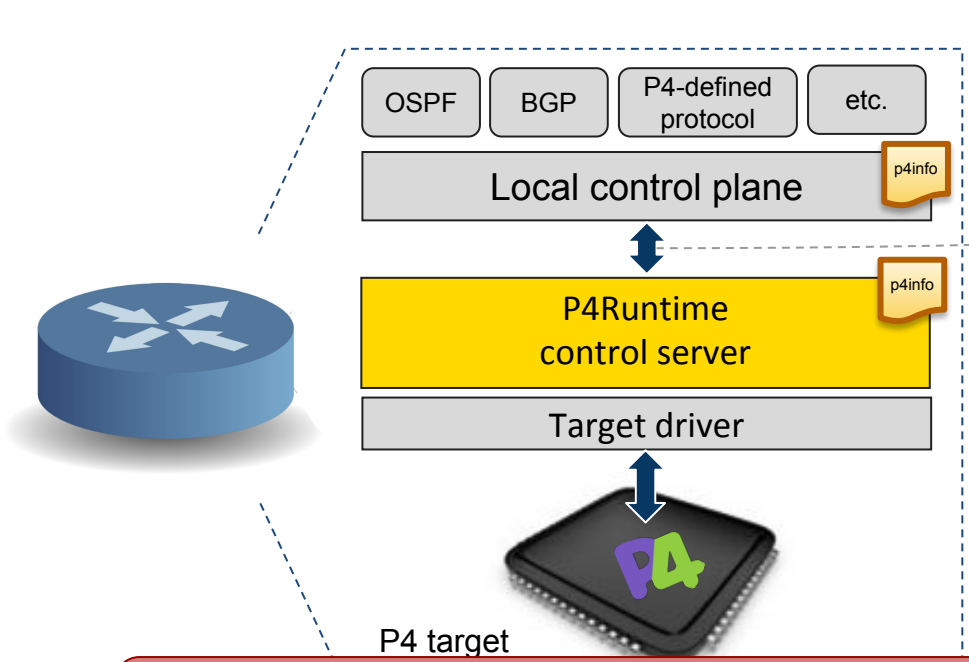
Protobuf message

```
table_entry {
  table_id: 33581985
  match {
    field_id: 1
    lpm {
      value: "\n\000\002\002"
      prefix_len: 32
    }
  }
  action {
    action_id: 16786453
    params {
      param_id: 1
      value: "\000\000\000\000\000\n"
    }
    params {
      param_id: 2
      value: "\000\007"
    }
  }
}
```

Remote control



Local control



Same target-independent protobuf format

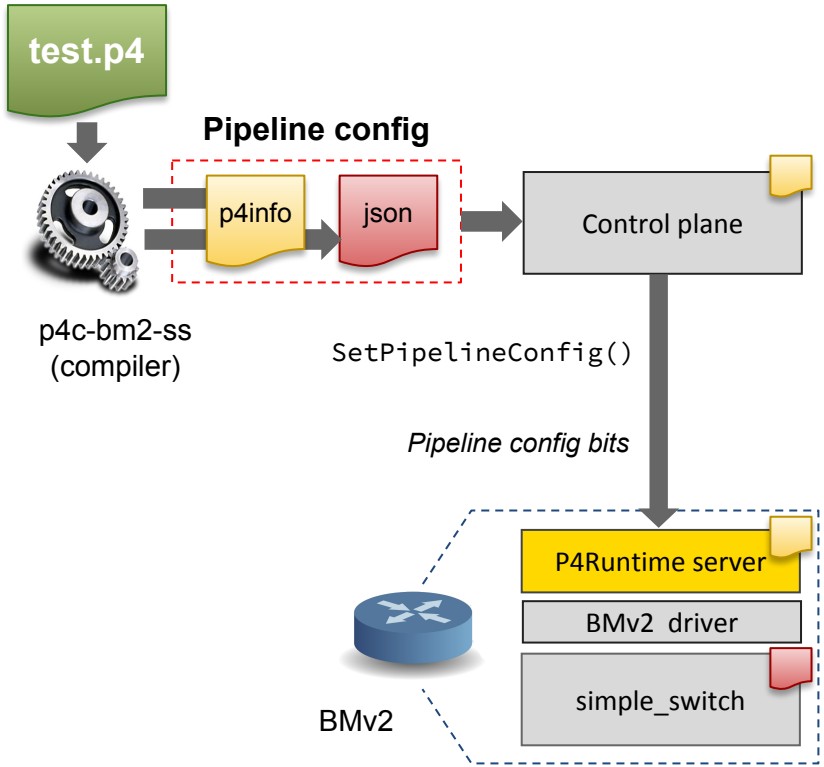
```
table_entry {
  table_id: 33581985
  match {
    field_id: 1
    lpm {
      value: "\f\000\...
      prefix_len: 8
    }
  }
  action {
    action_id: 16786453
    params {
      param_id: 1
      value: "\000\0...
    }
    params {
      param_id: 2
      value: 7
    }
  }
}
```

The P4 Runtime API can be used equally well by a remote or local control plane

Set Pipeline Config

p4runtime.proto simplified excerpt

```
message ForwardingPipelineConfig {
  P4Info p4info;
  /* Target-specific P4 configuration.
     e.g JSON bits for BMv2 */
  bytes p4_device_config;
  ...
}
```



P4Runtime API recap

Things we covered:

- **P4Info**
- **Table entries**
- **Set pipeline config**

What we didn't cover:

- **How to control other P4 entities**
  - Externs, counters, meters
- **Packet-in/out support**
- **Controller replication**
  - Via master-slave arbitration
- **Batched reads/writes**
- **Switch configuration**
  - Outside the P4 Runtime scope
  - Achieved with other mechanisms
    - e.g., OpenConfig and gNMI

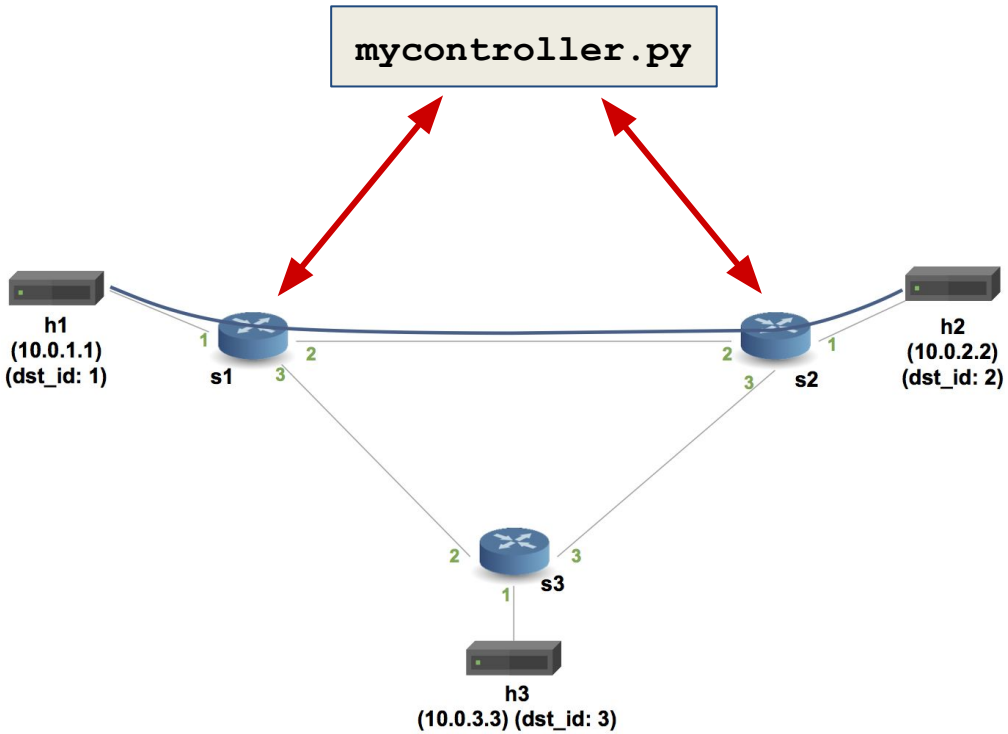
Work-in-progress by the p4.org API WG
Expect API changes in the future

P4 Runtime exercise

Exercise Overview

Controller's responsibilities:

1. **Establish a gRPC connection to the switches for the P4Runtime service**
2. **Push the P4 program to each switch**
3. **Write the tunnel forwarding rules:**
   a. `myTunnel_ingress` rule to encapsulate packets on the ingress switch
   b. `myTunnel_forward` rule to forward packets on the ingress switch
   c. `myTunnel_egress` rule to decapsulate and forward packets on the egress switch
4. **Read the tunnel ingress and egress counters every 2 seconds**



Getting started

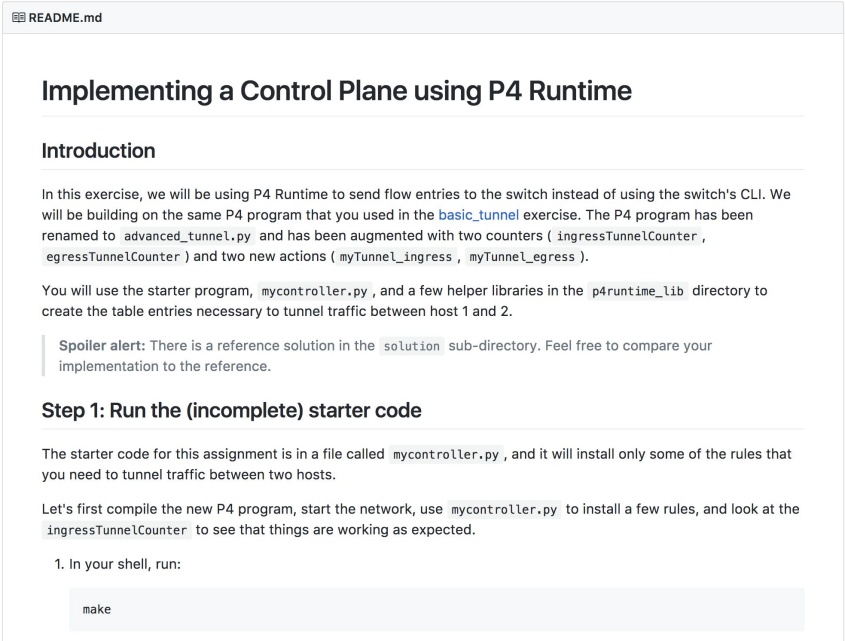
The source code has already been downloaded on your VM:
`~/tutorials/P4D2_2017_Fall/exercises/p4runtime`

You should start by reading the `README.md`

In this exercise, you will need to complete the implementation of `writeTunnelRules` in `mycontroller.py`

You will need two Terminal windows: one for your dataplane network (Mininet) that you will start using `make`, and the other is for your controller program.

To find the source code:
https://github.com/p4lang/tutorials/



README.md

Implementing a Control Plane using P4 Runtime

Introduction

In this exercise, we will be using P4 Runtime to send flow entries to the switch instead of using the switch's CLI. We will be building on the same P4 program that you used in the basic_tunnel exercise. The P4 program has been renamed to `advanced_tunnel.py` and has been augmented with two counters (`ingressTunnelCounter` , `egressTunnelCounter`) and two new actions (`myTunnel_ingress` , `myTunnel_egress`).

You will use the starter program, `mycontroller.py` , and a few helper libraries in the `p4runtime_lib` directory to create the table entries necessary to tunnel traffic between host 1 and 2.

> **Spoiler alert:** There is a reference solution in the `solution` sub-directory. Feel free to compare your implementation to the reference.

Step 1: Run the (incomplete) starter code

The starter code for this assignment is in a file called `mycontroller.py` , and it will install only some of the rules that you need to tunnel traffic between two hosts.

Let's first compile the new P4 program, start the network, use `mycontroller.py` to install a few rules, and look at the `ingressTunnelCounter` to see that things are working as expected.

1. In your shell, run:

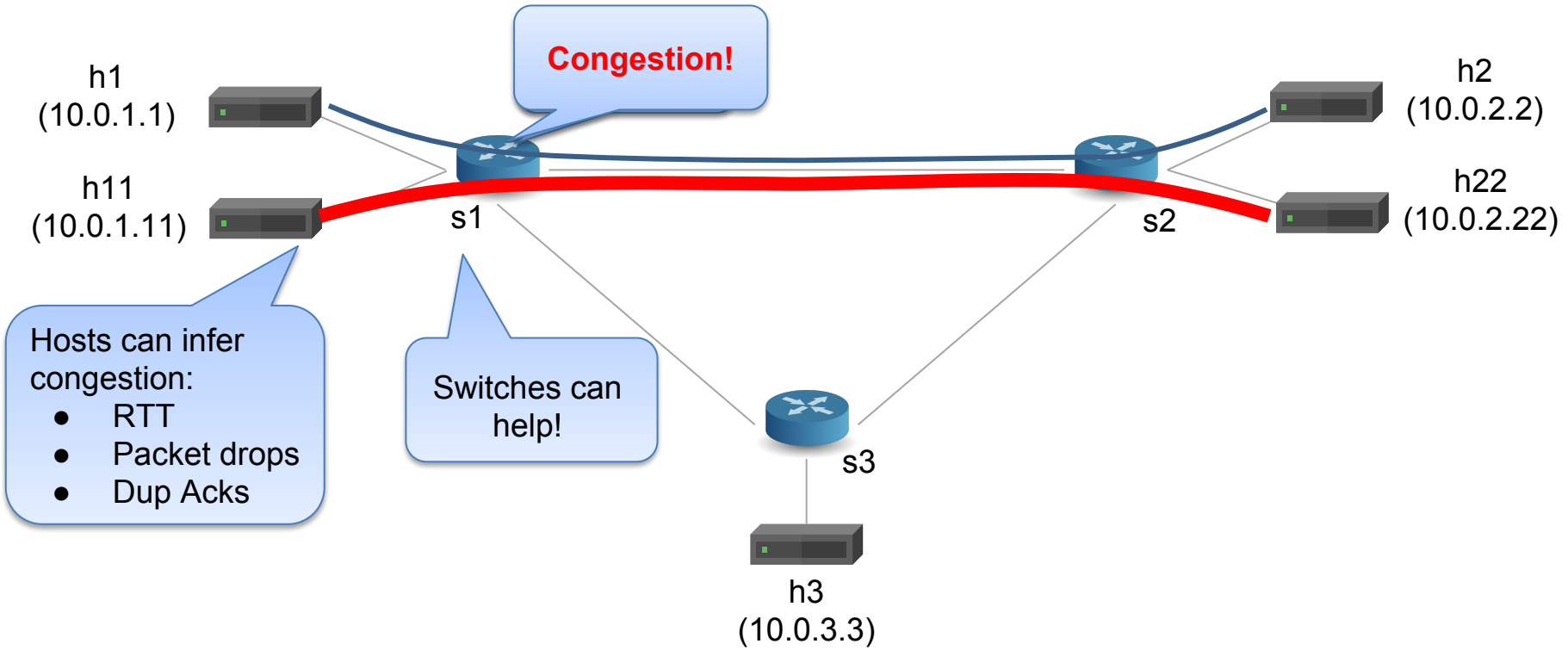
```
make
```

P4 Runtime takeaways

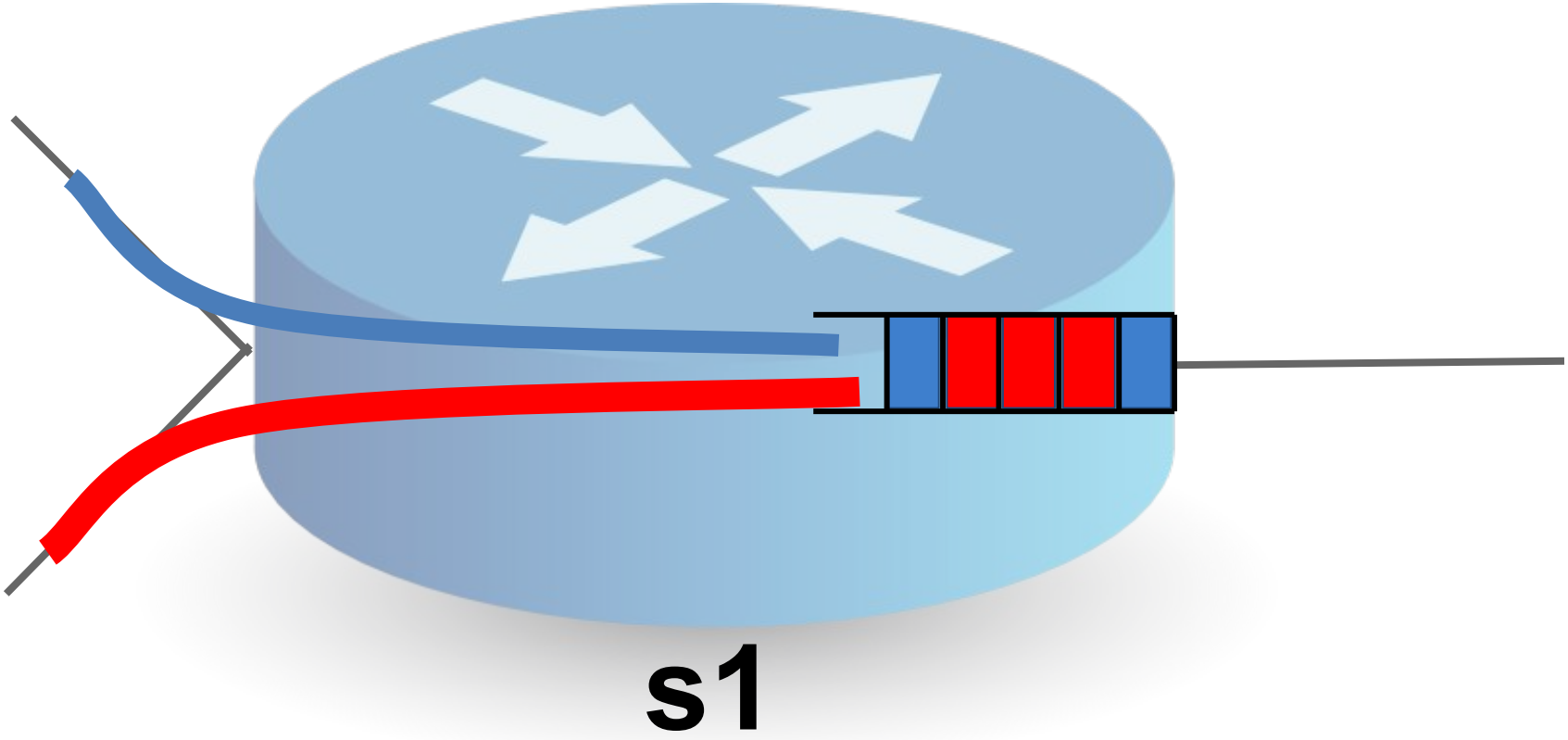
- **Program-independent API**
  - API doesn't change with the P4 program
  - No need to restart the control-plane with a different P4 program
- **Device does not need to be fully programmable**
  - Can be used on fixed-function devices
  - Provided that their behavior can be expressed in P4
- **Protobuf-based format**
  - Well-supported serialization format in many languages
  - Supported by many RPC frameworks, e.g., gRPC
    - Auto-generate client/server code for different languages
    - No need to define common RPC features (e.g., authentication)

Lab 3: Monitoring & Debugging

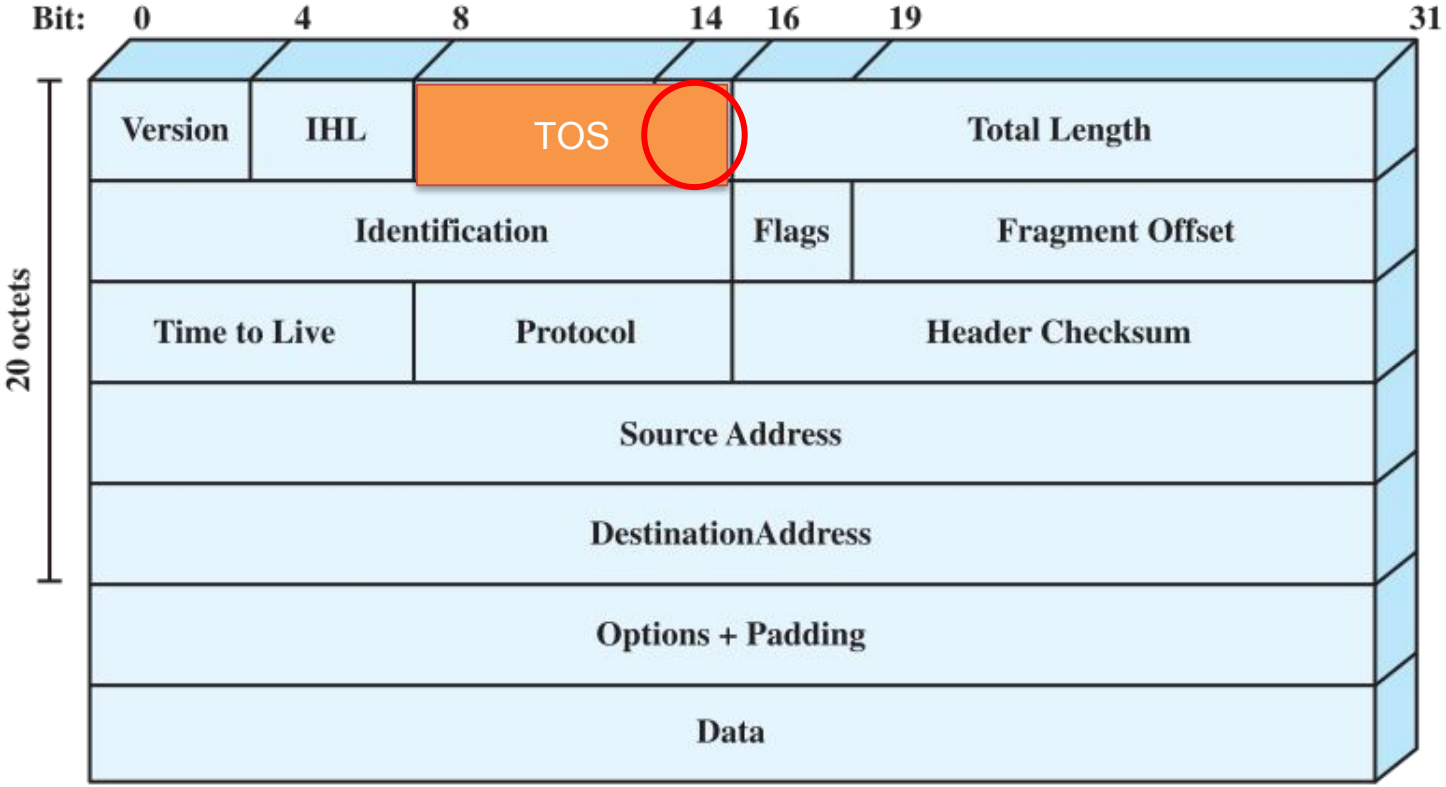
Monitoring & Debugging



Monitoring & Debugging



Explicit Congestion Notification



Explicit Congestion Notification

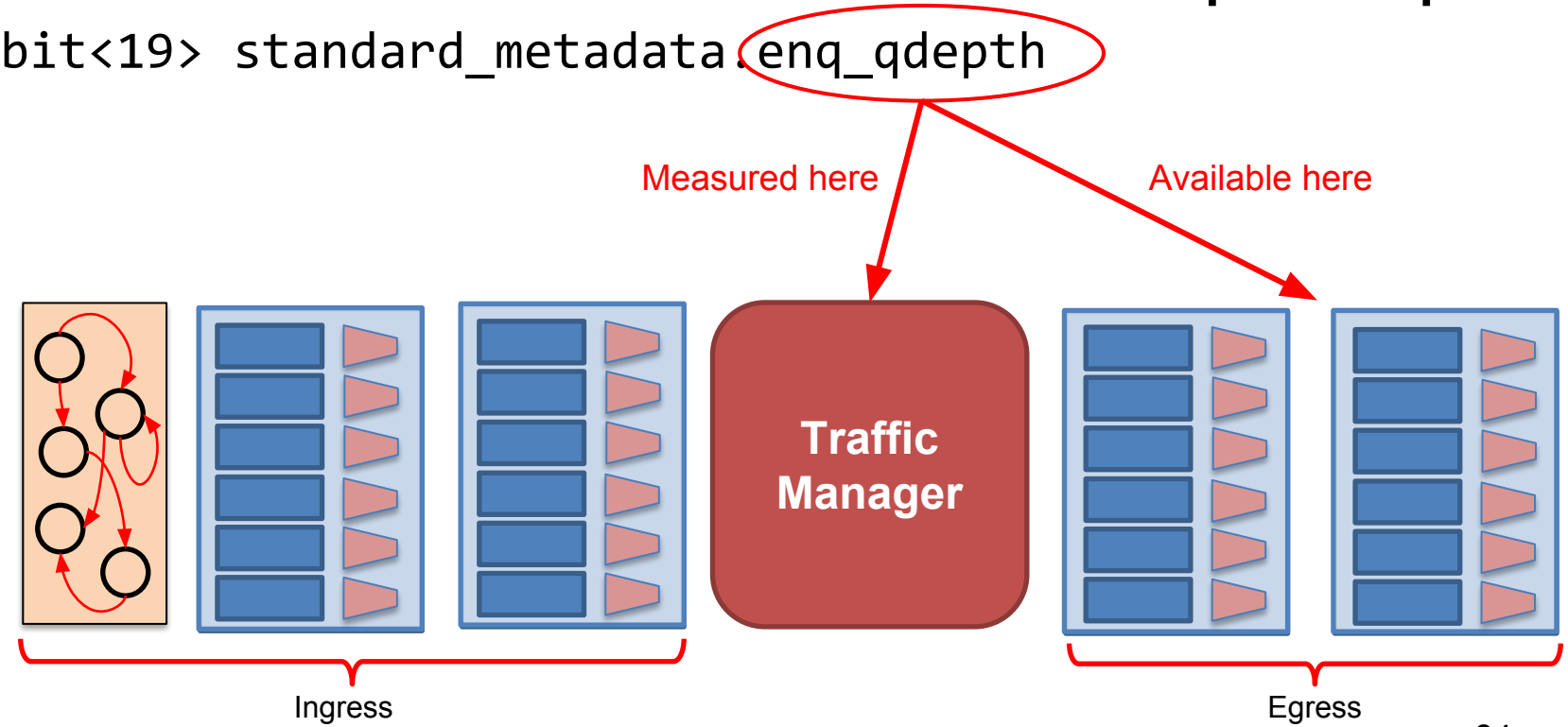
- **Explicit Congestion Notification**
  - `00`: Non ECN-Capable Transport, Non-ECT
  - `10`: ECN Capable Transport, ECT(0)
  - `01`: ECN Capable Transport, ECT(1)
  - `11`: Congestion Encountered, CE

- **For packets originating from ECT, ECN-capable switches set the CE bit upon congestion**
  - E.g., observed queue depth > threshold

Explicit Congestion Notification in P4

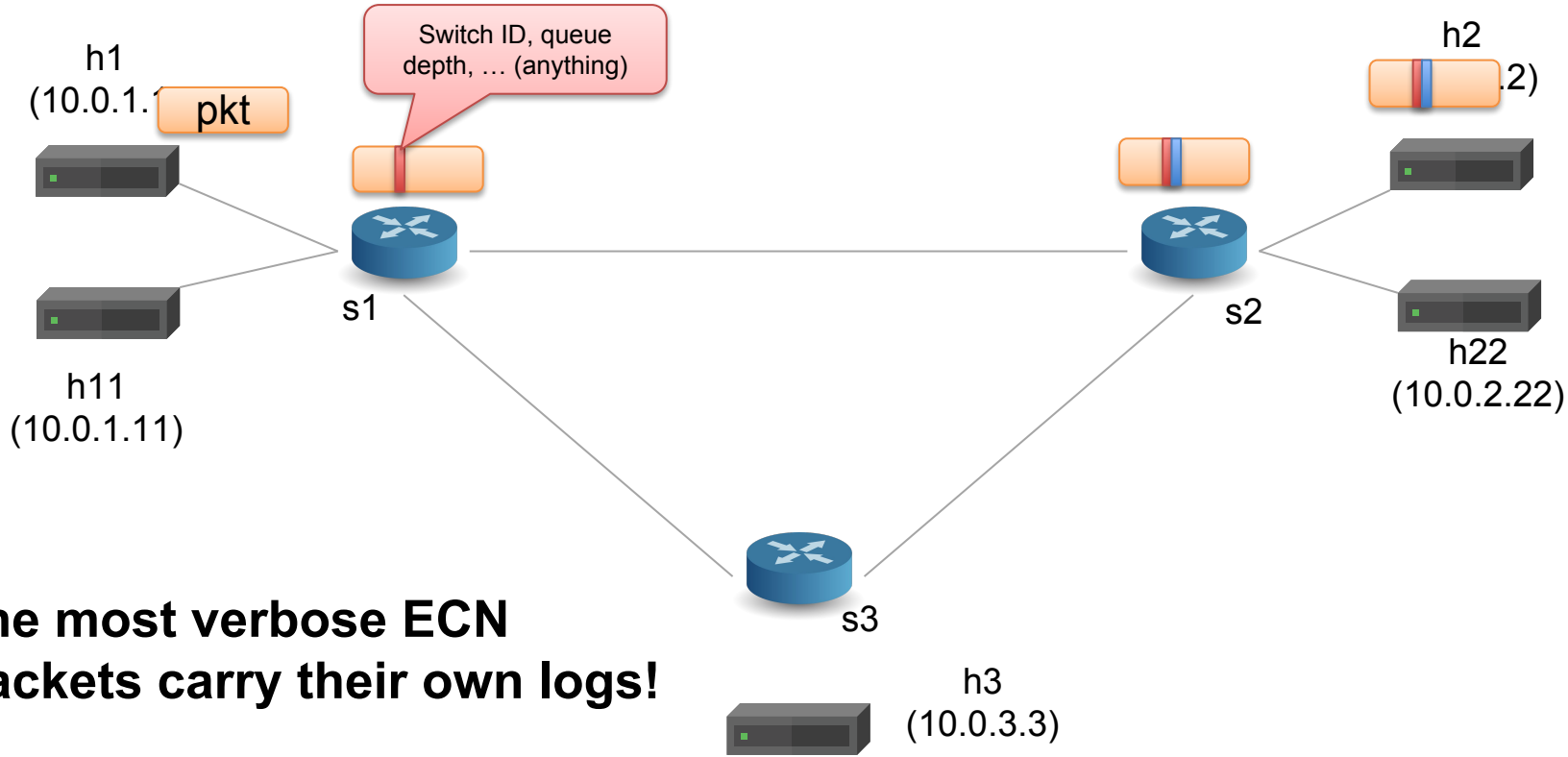
- **The standard data for the V1Model includes the queue depth:**

```
bit<19> standard_metadata.enq_qdepth
```



Coding Break

Multi-Route Inspection



The most verbose ECN
Packets carry their own logs!

Multi-Route Inspect: Packet Format

```
header mri_t {
  bit<16>  count;
}
```

```
header switch_t {
  switchID_t  swid;
  qdepth_t    qdepth;
}
```

```
struct headers {
  ethernet_t          ethernet;
  ipv4_t              ipv4;
  ipv4_option_t       ipv4_option;
  mri_t               mri;
  switch_t[MAX_HOPS]  swtraces;
}
```

- **Header validity opertions:**
  - `hdr.setValid()`: add_header
  - `hdr.setInvalid()`: remove_header
  - `hdr.isValid()`:test validity

- **Header Stacks**
  - `hdr[CNT] stk;`
- **Header Stacks in Parsers**
  - `stk.next`
  - `stk.last`
  - `stk.lastIndex`
- **Header Stacks in Controls**
  - `stk[i]`
  - `stk.size`
  - `stk.push_front(int count)`
  - `stk.pop_front(int count)`

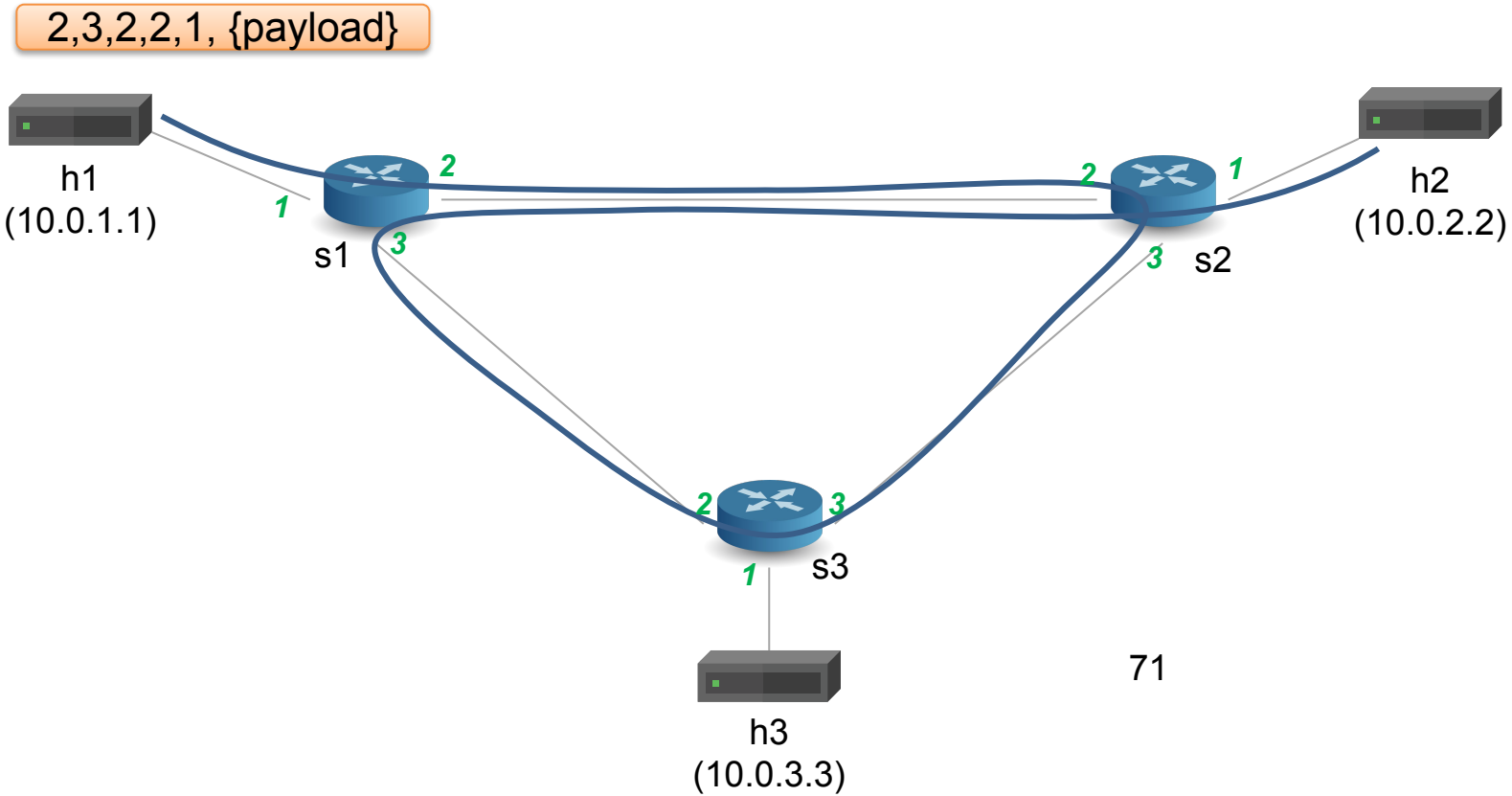
Header verification

```
/* Standard errors, defined in core.p4 */
error {
   NoError,           // no error
   PacketTooShort,    // not enough bits in packet for extract
   NoMatch,           // match expression has no matches
   StackOutOfBounds,  // reference to invalid element of a header stack
   OverwritingHeader, // one header is extracted twice
   HeaderTooShort,    // extracting too many bits in a varbit field
   ParserTimeout      // parser execution time limit exceeded
}
/* Additional error added by the programmer */
error { IPv4BadHeader }
...
state parse_ipv4 {
  packet.extract(hdr.ipv4);
  verify(hdr.ipv4.version == 4, error.IPv4BadHeader);
  transition accept;
}
```

Coding Break

Lab 4: Advanced Data Structures

Source Routing

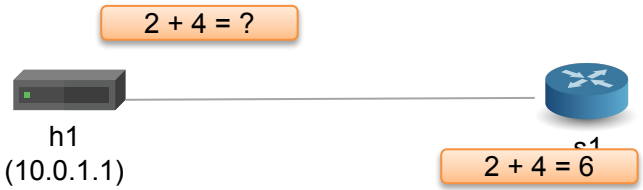


Source Routing: Packet Format

```
#define MAX_HOPS 9

const bit<16> TYPE_IPV4 = 0x800;
const bit<16> TYPE_SRCROUTING = 0x1234;
header srcRoute_t {
  bit<1>    bos;
  bit<15>   port;
}

struct headers {
  ethernet_t            ethernet;
  srcRoute_t[MAX_HOPS] srcRoutes;
  ipv4_t                ipv4;
}
```

- Parse source routes only if etherType is 0x1234
- The special value bos == 1 indicates the “bottom of stack”
- Forward packets using source routes, and also decrement IPv4 TTL
- Drop the packet if source routes are not valid

Calculator



Calculator: Packet Format

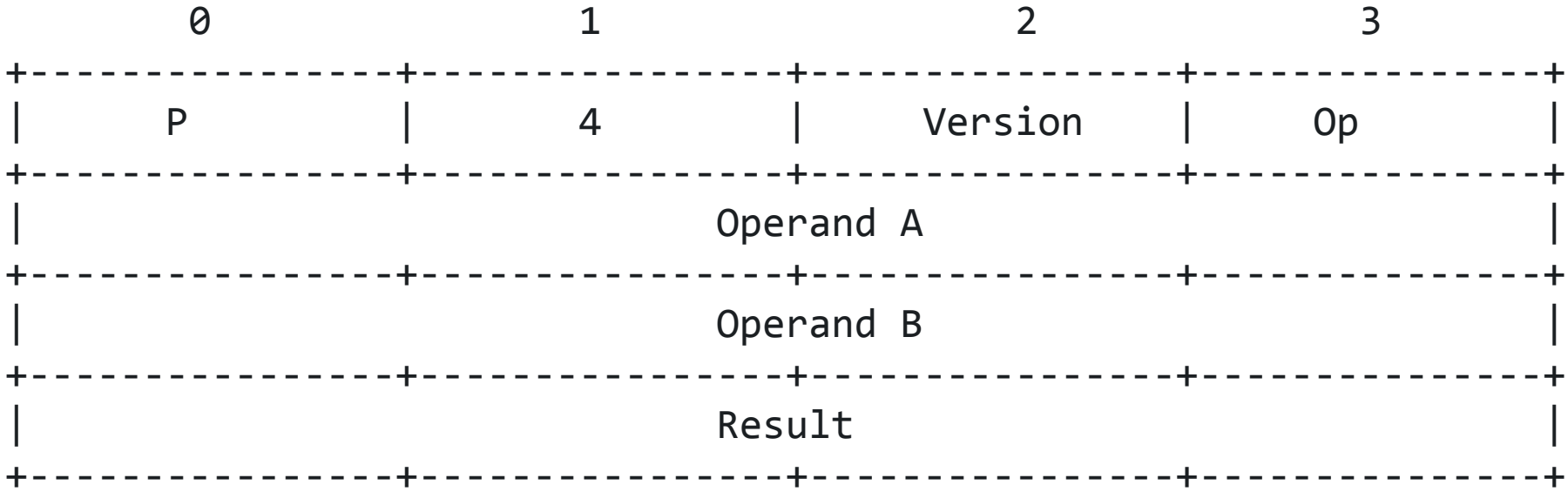


Table Initializers

```
table tbl {
  key = { hdr.h.f : exact }
  actions = { a1; a2; a3 }
  entries = {
   { 0x01 } : a1(1);
   { 0x02 } : a1(2);
   { _ } : NoAction();
  }
}
```

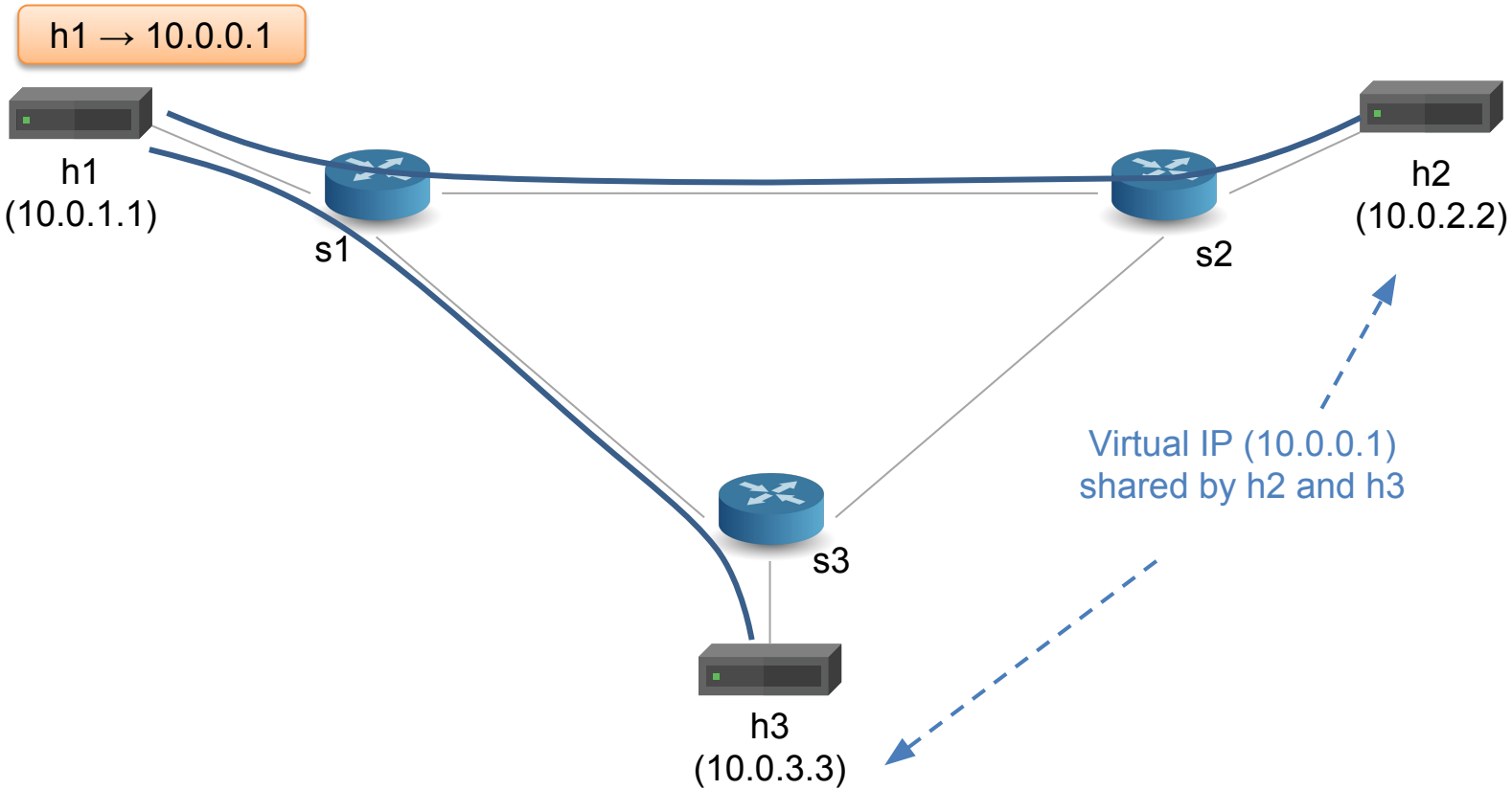
Can initialize tables with constant entries

Must fully specify the value of all action data, including values that are normally supplied by the control-plane

Hint: for the calculator, use a table that matches on the op-code

Lab 5: Dynamic Forwarding

Simple Load Balancing



Hashing (V1Model)

```
enum HashAlgorithm {
  csum16,
  xor16,
  crc32,
  crc32_custom,
  crc16,
  crc16_custom,
  random,
  identity
}
extern void hash<O, T, D, M>(
  out O result,
  in HashAlgorithm algo,
  in T base,
  in D data,
  in M max);
```

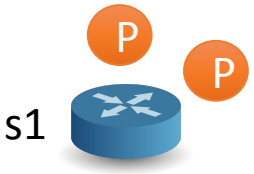
Computes the hash of `data` (using `algo`) modulo `max` and adds it to `base`

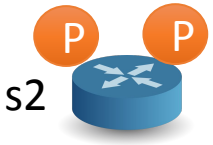
Uses type variables (like C++ templates / Java Generics) to allow hashing primitive to be used with many different types.

Dynamic Forwarding: HULA

1Mbps

1Mbps





Src ToR	Best path
s1	via s11
s2	via s11

h1
(10.0.1.1)

h2
(10.0.2.2)

h3
(10.0.3.3)

Dynamic Forwarding: HULA

1Mbps

1Mbps

Dst ToR	Best path
s3	via s11

Dst ToR	Best path
s3	via s11

Src ToR	Best path
s1	via s11
s2	via s11

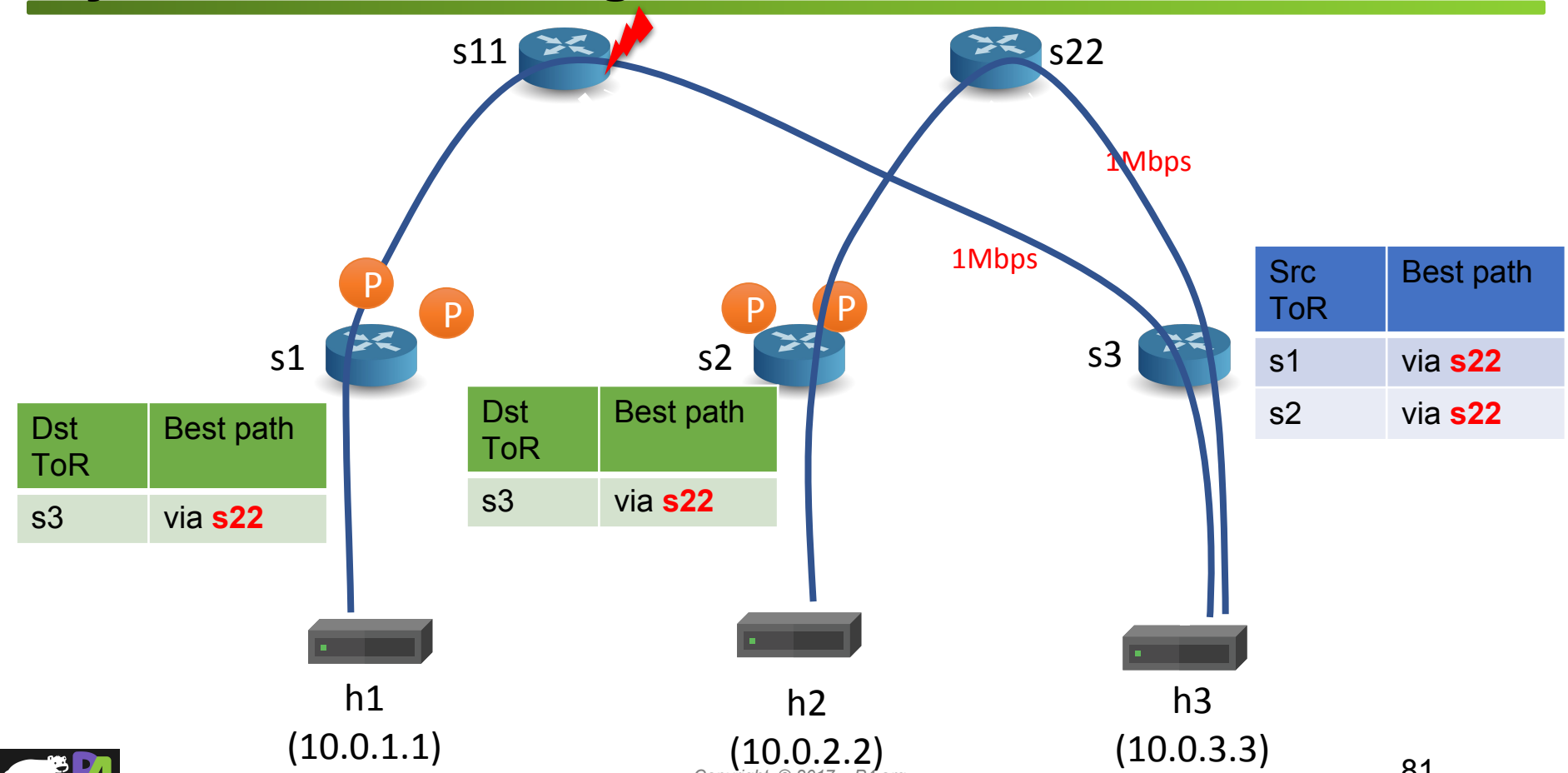
80

h1
(10.0.1.1)

h2
(10.0.2.2)

h3
(10.0.3.3)

Dynamic Forwarding: HULA



Dst ToR	Best path
s3	via **s22**

Dst ToR	Best path
s3	via **s22**

Src ToR	Best path
s1	via **s22**
s2	via **s22**

Registers (V1Model)

Definition

```
/* From v1model.p4 */

extern register<T> {
  register(bit<32> instance_count);
  void read(out T result, in bit<32> idx);
  void write(in bit<32> idx, in T value);
}
```

Usage (Calculating Inter-Packet Gap)

```
register<bit<48>>(16384) last_seen;

action get_inter_packet_gap(out bit<48> interval,
                            bit<14> flow_id) {
  bit<48> last_pkt_ts;
   /* Get the time previous packet was seen */
   last_seen.read((bit<32>)flow_id, last_pkt_ts);
   /* Calculate interval */
   interval = std_meta.ingress_global_timestamp –
              last_pkt_ts;
   /* Update the register with the new timestamp */
   last_seen.write((bit<32>)flow_id,
                   std_meta.ingress_global_timestamp);
}
```

Wrapping up & Next Steps

Why $P4_{16}$?

- **Clearly defined semantics**
  - You can describe what your data plane program is doing
- **Expressive**
  - Supports a wide range of architectures through standard methodology
- **High-level, Target-independent**
  - Uses conventional constructs
  - Compiler manages the resources and deals with the hardware
- **Type-safe**
  - Enforces good software design practices and eliminates “stupid” bugs
- **Agility**
  - High-speed networking devices become as flexible as any software
- **Insight**
  - Freely mixing packet headers and intermediate results

Things we covered

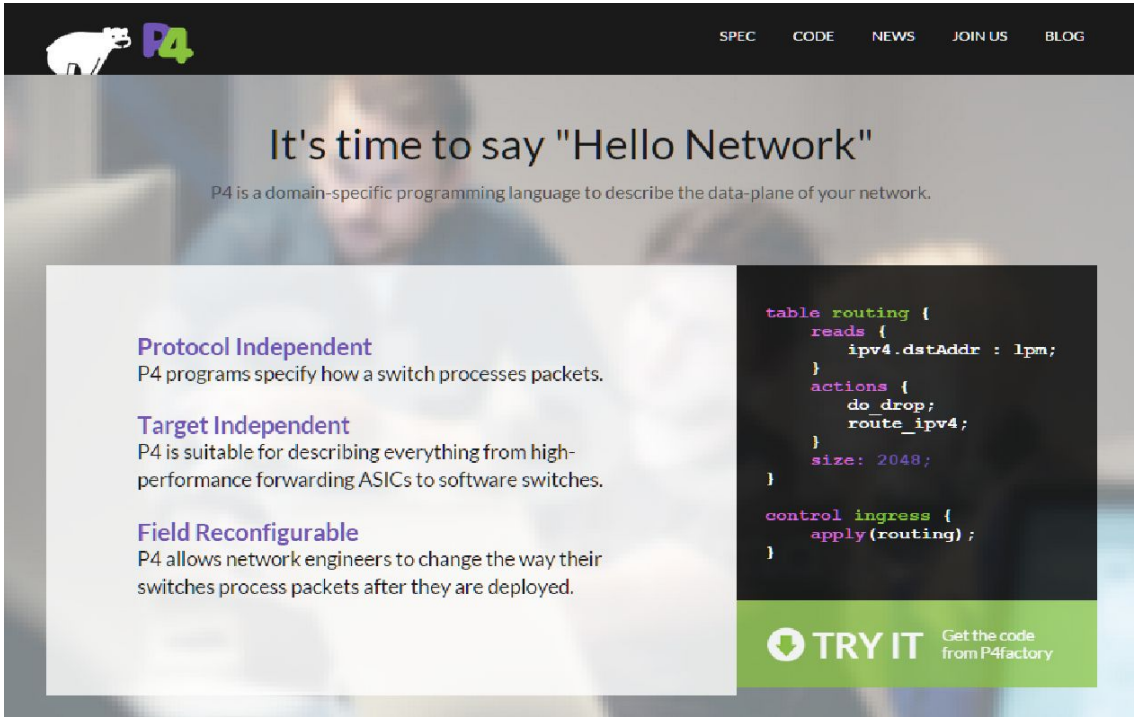
- **The P4 ”world view”**
  - Protocol-Independent Packet Processing
  - Language/Architecture Separation
  - If you can interface with it, it can be used
- **Key Data Types**
- **Constructs for packet parsing**
  - State machine-style programming
- **Constructs for packet processing**
  - Actions, tables and controls
- **Packet deparsing**
- **Architectures & Programs**

Things we didn't cover

- **Mechanisms for modularity**
  - Instantiating and invoking parsers or controls
- **Details of variable-length field processing**
  - Parsing and deparsing of options and TLVs
- **Architecture definition constructs**
  - How these "templated" definitions are created
- **Advanced features**
  - How to do learning, multicast, cloning, resubmitting
  - Header unions
- **Other architectures**
- **Control plane interface**

The P4 Language Consortium

- Consortium of academic and industry members
- Open source, evolving, domain-specific language
- Permissive Apache license, code on GitHub today
- Membership is free: contributions are welcome
- Independent, set up as a California nonprofit



P4.org Membership

- **Open source**, evolving, domain-specific language
- Permissive Apache license, code on GitHub today
- **Membership is free**: contributions are welcome
- Independent, set up as a California nonprofit

Thank you